LEVEL II

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED

AD A106722

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A106 722	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 VOICE FLOW CONTROL IN INTEGRATED PACKET NETWORKS. 12		5. TYPE OF REPORT & PERIOD COVERED ~~Thesis~~ 9 Doctoral thesis,
		6. PERFORMING ORG. REPORT NUMBER 14 LIDS-TH-1152
7. AUTHOR(s) 10 Howard Paul Hayden		8. CONTRACT OR GRANT NUMBER(s) ARPA Order [redacted] 3045/5-75 ONR N00014-75-C-1183, 15
9. PERFORMING ORGANIZATION NAME AND ADDRESS Massachusetts Institute of Technology Laboratory for Information and Decision Systems Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Code No. 5T10 ONR Identifying No. 049-383
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE 11 October 1981
		13. NUMBER OF PAGES 164 12 165
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Digitized Speech Computer Networks Flow Control

Voice and Data Networks

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Packet-switched networks are used primarily to handle data traffic, but considerable interest has recently been generated in extending packet-switching methods to also handle digitized voice traffic. New netwrok control procedures are needed to deal with packetized voice traffic, as its characteristics are different from regular data traffic.

We present two flow control algorithms which can be executed in a distributed manner to adjust source rates according to prevailing network conditions.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

410950

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

81 10 26 097

20. (Continued)

Although the algorithms were developed for packet-voice networks, they are quite general and are applicable to many other systems.

One algorithm is based on an optimization theoretic formulation of the flow control problem. The other has as its major premise a specific notion of fairness. Convergence is shown for both algorithms under static conditions.

A program is developed which simulates the behavior of a general packet-switched network on an individual packet basis. It is used to examine the performance of the second flow control algorithm under realistic conditions.

VOICE FLOW CONTROL IN INTEGRATED PACKET NETWORKS

by

Howard Paul Hayden
B.S.E.E.,George Washington University (1979)

Submitted in Partial Fulfillment
of the Requirements for the
Degrees of
Master of Science
and
Electrical Engineer

at the

Massachusetts Institute of Technology

June 1981

Signature of Author Howard Paul Hayden
Department of Electrical Engineering and Computer Science
May 15,1981

Certified by Pierre A. Humblet
Pierre A. Humblet, Thesis Supervisor

Accepted by ______________________
Arthur C. Smith, Chairman, Departmental Commitee on Graduate Students

VOICE FLOW CONTROL IN INTEGRATED PACKET NETWORKS

by

HOWARD PAUL HAYDEN

Submitted to the Department of Electrical Engineering and Computer Science on May 15, 1981 in partial fulfillment of the requirements for the Degrees of Master of Science and Electrical Engineer.

ABSTRACT

Packet-switched networks are used primarily to handle data traffic, but considerable interest has recently been generated in extending packet-switching methods to also handle digitized voice traffic. New network control procedures are needed to deal with packetized voice traffic, as its characteristics are different from regular data traffic.

We present two flow control algorithms which can be executed in a distributed manner to adjust source rates according ~~to~~ prevailing network conditions. Although the algorithms were developed for packet-voice networks, they are quite general and are applicable to many other systems.

One algorithm is based on an optimization theoretic formulation of the flow control problem. The other has as its major premise a specific notion of fairness. Convergence is shown for both algorithms under static conditions.

A program is developed which simulates the behavior of a general packet-switched network on an individual packet basis. It is used to examine the performance of the second flow control algorithm under realistic conditions.

Thesis Supervisor: Pierre A. Humblet

Title: Assistant Professor of Electrical Engineering

/or

al

A

ACKNOWLEDGEMENT

The author wishes to express his sincere appreciation to Professor Pierre Humblet for his complete guidance and encouragement during this work. Without his numerous contributions, completion of this thesis would have not been possible.

My office-mate Michael Hluchyj deserves special thanks for always taking time out from his busy schedule to answer many questions and to discuss various issues, quite frequently at great length.

I thank Mrs. F. Frolik for her skillful typing of this thesis.

My dearest thanks go to my parents for all their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

The development of packet-switching concepts has been quite rapid over the past several years. Numerous packet-switched data networks have been designed and implemented with much success. As a result, packet-switching has indeed proved to be a most cost-effective technique for handling variable and bursty traffic.

Conversational (real-time) speech is also bursty and thus methods for extending packet-switching concepts for voice have been receiving increasing attention. The performance criterion of voice networks are quite different than that of data networks and because of this fact network algorithms which have been developed for packet data traffic would yield unsatisfactory performance in the case of packet-voice traffic.

In this thesis we present two flow control algorithms for packet-switched networks which support traffic sources such as voice. However the algorithms are very general and not exclusively designed for voice traffic and thus can be applied to many other systems. A convergence proof is given for both algorithms and performance of one of the algorithms is evaluated through computer simulation.

1.1 Circuit-Switched vs Packet-Switched

Traditionally the design of communication networks for continuous traffic sources (e.g. voice, data file transfers) has been based upon the concept of circuit-switching. In circuit-switched networks when a conversation between two terminals is initiated, an end-to-end circuit is established for the pair of users. The end-to-end transmission facilities are then dedicated to the users until either party hangs up, whereupon the circuit is disconnected. The most familiar example of such a network is the common carrier telephone network.

Interactive computer-to-computer data transactions tend to be bursty in nature and thus dedication of network resources to each user would result in a high level of inefficiency. This observation has given rise to the development of packet-switching methods for data transactions. A packet-switched network may be thought of as a distributed pool of resources (channels, buffers, and switching processors) whose capacity must be shared dynamically by a community of competing users wishing to communicate with each other. Thus in packet-switched networks each user dynamically shares network resources by using them only when information is being sent.

In a normal voice conversation a speaker is active about 50% of the time, and thus voice conversations using traditional circuit-switched networks are wasting about 50% of the network resources. This fact was noted by telephone system engineers and during the past two decades several techniques have been developed to take advantage of the so-called "talkspurt/silence"phenomenon associated with conversational speech.

The earliest strategy was the Bell System Time Assignment Speech Interpolation (TASI) [1] used on intercontinental voice connections in which channel capacity is allocated only when appropriate hardware detected that a subscriber was actively speaking. Once the channel is seized, the speaker is given uninterrupted access to the channel. During periods of silence, the channel is relinquished and becomes available to other speakers. Digital variations of the original TASI concept, such as Digital Speech Interpolation (DSI) [2], and Speech Predictive Encoding [3] have also been implemented.

Recently considerable interest has been generated [4], [5], in the use of packet-switching methods for simultaneously handling voice and data traffic in integrated digital communication networks. There are many features of an all-packet integrated system. Some of the features are:

(1) switch economies can result since facilities for storing, forwarding, and routing packets can be basically identical for both traffic categories

(2) the capability to accommodate new applications which must access different types of data or voice processes

(3) since conversational (real-time) speech is bursty in nature the packet-switching concept allows one to exploit this fact by forming and transmitting packets only during periods of actual speaker activity.

The latter feature is actually the major impetus for the development of packet-switched voice networks since it affords a convenient and powerful mechanism for extending the TASI technique discussed previously to multilink network configurations.

1.2 Flow Control

Packet-switching offers many advantages, the primary ones being greater speed and flexibility in setting up user connections across the network and more efficient use of network resources after the connection is established. Unfortunately these advantages do not come without a certain danger. Unless careful control is exercised on the user demands, the users may seriously abuse the network. In fact, if the demands are allowed to exceed the system capacity, highly unpleasant congestion effects occur which rapidly neutralize the efficiency advantages of a packet network by increasing delay. Thus networks cannot afford to accept all the traffic that is offered to them without control. There must be rules which govern the acceptance of traffic flow from outside the network and coordinate flow inside the network. These rules are commonly known as "flow control procedures". More precisely flow control is the set of mechanisms whereby a flow of traffic can be maintained within limits compatible with the amount of available network resources.

Flow control techniques have evolved almost independently in circuit-switched voice systems and packet-switched data networks due primarily to the differences in source characteristics and service requirements for the two traffic classes.

In recent years flow control strategies for data-only packet networks have been developed to a significant level of sophistication (the reader is referred to Gerla and Kleinrock [6] for a very comprehensive and up-to-date overview of packet-data flow control methodologies). The basic approach is to somehow curtail the rate of traffic generation under heavy network traffic conditions. This prevents excessive queueing delays from developing at internal switching facilities.

There has been little systematic work done in the area of voice flow control. Currently the most commonly used technique for voice flow control in circuit-switched networks is simply call blocking, i.e. preventing the initiation of new calls during busy periods. The TASI type systems are a simple form of voice flow control because they "freeze-out" speakers when the number of active speakers temporarily exceeds the available channel capacity. This "cutout" phenomenon results in clippings and segmentation of certain conversations with an associated loss in intelligibility. Refinement of the TASI concept based on digital encoding techniques have been developed and implemented whereby the bandwidth per active speaker is systematically reduced to accommodate additional speakers.

In developing flow control stategies for packet-voice networks one should not simply apply the techniques developed for packet-data networks because the main performance objectives for the two traffic classes are opposite. The criterion for data is integrity (i.e. no errors) with delay being a secondary consideration. Wherease voice communications like many other real-time applications, is better off with low delay even at the expense of reduced quality.

Thus a reasonable flow control objective for voice is one in which delay remains small while speech quality is dynamically traded in response to network traffic variations. In circuit-switched voice networks simple voice flow control can be applied at dial-up time by defining a bit rate at which the converation will be carried out. In its extreme case this reduces to call blocking, i.e., to an assigned bit rate of zero; in the more general case it could imply that two speakers engaged in point-to-point conversation are each assigned different bit rates due to separate routing of their streams or because network congestion is not directionally symmetrical between given nodes [7].

A more dynamic approach to voice flow control can be evolved by allowing bit rates to change during actual conversation. Very little work has been done in this area for circuit-switched voice systems, partly because the scheme is poorly matched to the notion of a fixed capacity assignment for a given voice stream. One can do better using packet-switched networks since the speech quality of each source can be easily varied by utilizing speech encoding techniques and by discarding a small percentage of its voice packets. Furthermore instanteneous voice link overloads can be alleviated by appropriate queueing and buffering actions.

1.3 Previous Work

At present there has been only one effort known to this author, to develop a dynamic packet voice flow control scheme. This effort was initiated by M.I.T. - Lincoln Laboratory (LL). The Lincoln Laboratory flow control scheme is based upon a speech digitization concept called "embedded coding" first proposed by the Naval Research Laboratory [8], [9]. In brief one encodes speech into a set of priority-ranked packets such that if all are received, the result is high quality, high bit rate voice output. If lower priority packets are not delivered, the synthesizer can still use the received high priority subset to produce speech at a lower but still useable bit rate.

Although embedded packetization by itself does not constitute a flow control strategy, it affords a mechanism by which voice rates can be adjusted downward by network switches on a packet-by-packet basis without waiting for control messages, etc., to propagate through the network to various voice terminals. In effect, it permits intermediate nodes to instantly reduce the bit rates of voice conversations as needed, by discarding lower priority packets, without total loss of communication. This capability allows one to initiate global flow control strategies for dealing with source coder rates. However, one should note that if there are bottleneck links in the network this scheme would lead to inefficiency since a large amount of network resources would have been spent on traffic which gets discarded.

An end-to-end voice bit rate control technique has been suggested for use in conjunction with an embedded-coding packet network [10]. In brief voice conversations are conducted over fixed routes. Traffic overloads at intermediate nodes are handled by discarding voice packets of lower priority which tends to lower the bit rates at which users communicate. Each voice terminal reports the bit rate at which it is receiving speech traffic to its companion terminal across the network. Transmitting encoders respond to this information by appropriately discarding packets before they enter the network, thereby matching their rates to prevailing network conditions. Provision is made to allow rates to increase when network links are lightly loaded.

The performances of the Lincoln Laboratory flow control schemes were evaluated by a computer simulation program [11]. The simulation used a model network which consists of a central node through which pass 16 paths connecting four nodes on either side of the central node. The model provides two hops from source to destination along with competition for resources at the central node.

The simulation was performed for three separate cases, each case incorporating a different type of end-to-end feedback control strategy. In general, feedback control systems have the potential for unstable behavior (i.e. oscillations), and in fact two of the four L.L. simulation experiments have shown that extreme temporal variations in received speech can result from inappropriate feedback control dynamics.

The third L.L. end-to-end flow control scheme, the so-called "phantom-probe" strategy, resulted in constant steady state received bit rates when tested under the same network conditions. Although the phantom-probe strategy appears to achieve the desired goal of developing a flow control strategy which maintains stable operation, it is by no means clear whether this behavior would be achievable in more general large scale networks.

1.4 Motivation

Our attention was first drawn to the area of voice flow control after examining the L.L. flow control scheme and its corresponding simulation results. Our preliminary research work was to develop and analyze a simplified, analytical model of the L.L. flow control scheme. Since the objective of our investigation was to yield some general results, we decided to focus our attention on the end-to-end control process rather than the effects of the "packet-stripping" operation. Hence, we neglected the capability of discarding packets at intermediate nodes in the network (i.e. local control) and assumed that control was carried out by only allowing source rates to change in accordance with received feedback reports. We further assumed that all sources were deterministic (i.e. the rate of all sources were dictated by the state equations which govern the evolution of the system) and that all critical delays were known precisely.

In brief our findings showed that in order to insure prevention of received rate oscillations, the end-to-end feedback control scheme must properly take into account the presence of delay between the time control decisions are made and the time they take effect.

Although the original objective of our investigation was not to develop a stable end-to-end flow control scheme, the results of our investigation did however yield sufficient knowledge to properly do so. Basically, it was this new insight and the fact that no known analytical work had been done in the area of packetized voice flow control, which supplied us with the necessary impetus to initiate our research effort.

1.5 Problem Description

Equipped with our important preliminary findings our goal is to systematically develop a stable and robust adaptive voice flow control scheme which would be both analytically and practically sound.

The main functions of the flow control scheme shall be:

(1) prevention of excessive delays due to overloads

(2) fair and efficient allocation of resources among competing users.

Unfortunately the efficiency and fairness objectives do not always coincide. One of the functions of flow control, therefore, is to prevent unfairness by placing selective restrictions on the amount of resources that each user may acquire, in spite of the negative effect these restrictions may have on network efficiency. In addition, the fairness criteria is one which is highly debatable and thus adequate evaluation of how a particular fairness policy interacts with a specific flow control scheme must be carried out in order to establish the strategy which achieves the "best" overall network performance.

In order for any flow control scheme to be readily adaptable to current existing packet-networks, it should not place any severe requirements on the network structure. For example, the L.L. flow control scheme, which requires specific packet-stripping operations to be performed at every network link suffers "loss of modularity", because it demands that the network be matched to the peculiarities of the digitizers. As a result, we have decided at this time to carry out our control by adaptively adjusting the source rates in accordance with the prevailing network conditions, only at the gates of the network. We assume that voice quality will vary monotonically with source rate, where in general, the higher the source rate, the better the voice quality.

Thus, the overall objective of the control scheme is to allow a graceful, and fair degradation of user service as network traffic increases, and a fair improvement of user service as traffic decreases.

1.6 Thesis Outline

The goal of this thesis is to develop flow control algorithms which achieve the desired objectives described in section 1.5.

In chapter II we first specify our network model and formulate the problem mathematically.

Using the tools of optimization theory, in chapter III we view flow control as an optimization problem. After a general discussion of distributed algorithms we present a decentralized algorithm to solve the optimization and a proof of its convergence.

A flow control algorithm based on a notion of "fairness" coincident with the standard voice communication network policy is first motivated and then developed in detail in chapter IV. A distributed version of the algorithm is given and is shown to always converge to the unique optimal solution. Possible extensions to the original scheme are also discussed.

In chapter V a detailed computer simulation program suitable for general packet-switched networks is first developed. This program is then used to examine the performance of our flow control algorithms presented in chapter IV. Results for several cases are examined.

CHAPTER II

GENERAL FORMULATION

In this chapter we formulate the flow control problem for a store and forward packet-switching network. However, our formulation is quite general and can be used for the design of flow control strategies in other types of communication networks. After the model is discussed, we present the mathematical formulation of the flow control problem.

2.1 The Network Model

Consider a store and forward packet-switching network. The network traffic is voice in steady exchange between users. We will refer to those users who are engaged in a conversation as active, all others will be referred to as inactive. In general as time proceeds some of the active users may become inactive and some of the inactive users may become active. Although we are primarily interested in developing flow control schemes for a quasi-static situation, that is a situation whereby the users' requirements change slowly in time, for the purpose of theoretical development of the problem we consider a static case.

The static case assumes that all active users are always active and all inactive users are always inactive. Furthermore we assume that each active user always has some information he wishes to transmit. Clearly the proceeding assumptions allow us to view the behavior of all network users as being deterministic in the sense that there is no uncertainty associated with the message arrival process. In addition inactive users will be of no concern to us in our development and thus we shall not consider them in our model.

Let use assume that the store/forward packet switching network consists of M active users and N communication links.

Let $\mathcal{U}$ denote the set of all users in the network:

$$\mathcal{U} = \{u_i \mid i = 1,\ldots,M\} .$$

Let L denote the set of all links in the network:

$$L = \{j \mid j = 1,\ldots,N\} .$$

Let r_i denote the rate in (bits/sec) at which user u_i transmits information.

In vector form:

$$\vec{R} \triangleq \begin{bmatrix} r_1 \\ \cdot \\ \cdot \\ r_M \end{bmatrix}$$

For the purpose of accomplishing flow control in the network we assume that it is somehow possible for each user u_i to set its rate to the value which is determined by the flow control algorithm. The actual practical method for accomplishing this will be discussed later.

Now let f_j denote the flow of traffic in (bits/sec) on link j. In vector form:

$$\vec{F} \triangleq \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix}$$

We assume that each network user utilizes a single fixed route to carry out its conversation. Due to our previous static assumptions we note that the contribution of traffic flow due to a particular user, say u_i, is equal on all links which user u_i utilizes in his route. Furthermore, the value of this flow contribution is precisely equal to the rate of user u_i, namely r_i.

To complete our network model we introduce a link-user incidence matrix, denoted H. The purpose of the H matrix is simply to describe in a convenient form the set of links which each user utilizes.

Let

$$H \triangleq \begin{array}{c} 1 \\ i \\ N \end{array} \begin{bmatrix} h_{11} & & h_{1M} \\ & h_{ij} & \\ h_{N1} & & h_{NM} \end{bmatrix}$$

where

$$h_{ij} = \begin{cases} 1 & \text{if user } j \text{ utilizes link } i \\ 0 & \text{otherwise} \end{cases}$$

At this pointour preceding assumptions and definitions allow us to state the following relationship:

$$f_j = \sum_{i=1}^{M} h_{ji}\, r_i \qquad 1 \leq j \leq N \tag{2.1}$$

or in vector form;

$$\vec{F} = H\vec{R} \tag{2.2}$$

In practice the maximum allowable traffic flow on a link is limited due to physical constraints. However for theoretical development we will view the goal of flow control as determining optimum (in some specified sense) user rates, which allow each link i to be utilized up to some value smaller than the capacity which we will call the effective link capacity, denoted c_i. If we admit that the algorithm will effectively maintain the flow close to the desired maximum (say 0.8 of the true capacity) we can assume that the true capacity is infinite.

Thus if r_i^* and f_j^* denote the optimum user rate assignments and resultant link flows respectively, we must have:

$$f_j^* \leq c_j \qquad 1 \leq j \leq N \qquad (2.3)$$

or equivalently using (2.1),

$$\sum_{i=1}^{M} h_{ji}\, r_i^* \leq c_j \qquad 1 \leq j \leq N \qquad (2.4)$$

2.2 Mathematical Formulation

In order to mathematically formulate the flow control problem we must first select an appropriate network objective function. A reasonable objective function is one which takes into account the satisfaction of the network users. Clearly the higher the rate at which a user is allowed transmit the more satisfied he is with the quality of service which he is receiving. With this objective in mind, for each user u_i we create a reward function denoted $e(r_i)$, which is an increasing function of the rate r_i allocated to user u_i. A typical function is shown in Fig. 2.1.

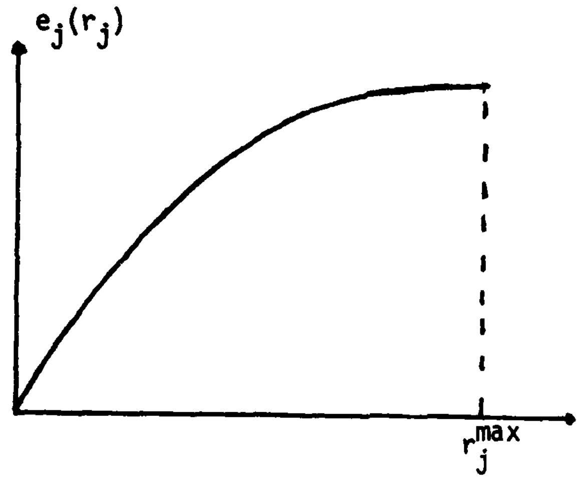


Figure 2.1 Typical User Reward Function

Note that the convex $\cap$ shape of the curve in Fig. 2.1 is a reasonable model of user satisfaction since one would expect that there is less to be gained by allocating additional rate to a user which already benefits from a high rate. In addition r_i^{Max} is the maximum rate at which user u_i would ever wish transmit.

Formally we have the following definition.

Definition 2.1 For each user of the network u_i, there is a reward function $e_i(r_i)$ assigned with the following properties:

(i) $e_i(r_i)$ is increasing on $[0,r_i^{Max}]$

(ii) $e_i(r_i)$ is convex $\cap$ on $[0,r_i^{Max}]$

We now form an aggregate network reward function as;

$$E(\vec{R}) = T\{e_j(r_j)\} \tag{2.5}$$

where T{x} is some specified function, which operates on the set {x} .

In the chapters which follow we will consider the development of flow control algorithms to maximize two different network objective functions subject to the constraint (2.4).

CHAPTER III

OPTIMIZATION THEORY APPROACH

Our primary goal in this chapter is to show that the flow control problem can be formulated and solved as a convex optimization problem. It is similar to the work of Golestaani [15], except that the routing problem is ignored, and the constraints are different.

In the first two sections we formulate the problem and determine the optimality conditions. Next we discuss distributed network algorithms within a general context and then develop the distributed flow control algorithm.

Finally, we show that convergence is guaranteed under certain conditions and discuss the implications of those conditions.

3.1 Problem Statement

In this chapter we choose as our network flow control objective function simply the sum of the individual user reward functions. The rationale for choosing this cost function is to see whether its simplicity leads to an efficient and easily implementable optimization algorithm. Thus we let $T\{x\} = \sum x_i$ in (2.5) and hence

$$E(\vec{R}) = \sum_{j=1}^{M} e_j(r_j) \tag{3.1}$$

Note that $E(\vec{R})$ is convex$\cap$ in $\vec{R}$. Now using (2.3) we can formulate the convex optimization problem as

$$\operatorname*{Max}_{\{r_i\}} E(\vec{R}) \tag{3.2a}$$

s.t.

$$f_i \leq c_i \qquad 1 \leq i \leq N \tag{3.2b}$$

$$0 \leq r_j \leq r_j^{Max} \qquad 1 \leq j \leq M \tag{3.2c}$$

Since our optimization variables are the $\{r_i\}$ we restate constraint (3.2b) in terms of the $\{r_i\}$ by using (2.4) and as a result we have the following equivalent formulation:

$$\operatorname*{Max}_{\{r_i\}} E(\vec{R}) \tag{3.3a}$$

s.t.

$$\sum_{j=1}^{M} h_{ij} r_j \leq c_i \qquad 1 \leq i \leq N \tag{3.3b}$$

$$0 \leq r_j \leq r_j^{Max} \qquad 1 \leq j \leq M \tag{3.3c}$$

3.2 Optimality Conditions

Theorem 3.1 The necessary and sufficient conditions on $\{r_i^*\}$ for the solution of (3.3) is that a set of non-negative numbers $\{\lambda_i^*\}$ $1 \leq i \leq N$ exist such that

for $1 \leq j \leq M$

$$\frac{de_j(r_j^*)}{dr_j} - \sum_{i=1}^{M} \lambda_i^* h_{ij} \left\{ \begin{array}{lll} = 0 & \text{if} & 0 < r_j^* < r_j^{Max} \\ \leq 0 & \text{if} & r_j^* = 0 \\ \geq 0 & \text{if} & r_j^* = r_j^{Max} \end{array} \right\} \qquad (3.4a)$$

for $1 \leq i \leq N$

$$\lambda_i^* \left[h_{ij} r_j^* - c_i\right]^- = 0 \qquad (3.4b)$$

Proof of Theorem 3.1

$E(\vec{R})$ is a concave function defined over a convex set $X \subset \mathbb{R}^N$ thus the set $X^* \subset X$ where $E(\vec{R})$ achieves a maximum is also a convex set. Furthermore every local maximum of $E(\vec{R})$ over X is a global maximum. Using the preceding observation and the fact that (3.3b) and (3.3c) are linear constraints we have the following proposition.

Proposition 3.1

The necessary and sufficient conditions on a feasible $\vec{R}^*$ to maximize $E(\vec{R})$ is that there exist unique vectors:

$$\vec{\lambda}^* = \begin{bmatrix} \lambda_1^* \\ \\ \lambda_N^* \end{bmatrix} \quad ; \quad \vec{\lambda}^* \in \mathbb{R}^N$$

$$\vec{U}^* = \begin{bmatrix} u_1^* \\ \\ u_M^* \end{bmatrix}; \qquad \vec{U}^* \in \mathbb{R}^N$$

and

$$\vec{\gamma}^* = \begin{bmatrix} \gamma_1^* \\ \\ \gamma_N^* \end{bmatrix}; \qquad \vec{\gamma}^* \in \mathbb{R}^N$$

Such that

$$-\frac{\partial E(\vec{R}^*)}{\partial r_j} + \sum_{i=1}^{M} \lambda_i^* h_{ij} - \gamma_j^* + u_j^* = 0 \qquad 1 \le j \le M \qquad (3.5a)$$

$$\lambda_i^* \ge 0 \;, \quad \lambda_i^* \left[\sum_{i=1}^{M} h_{ij} r_j^* - c_i\right] = 0 \qquad 1 \le i \le N \qquad (3.5b)$$

$$\gamma_j^* \ge 0 \;, \quad \gamma_j^* r_j^* = 0 \qquad 1 \le j \le M \qquad (3.5c)$$

$$u_j^* \ge 0 \;, \quad u_j^*(r_j^* - r_j^{Max}) = 0 \qquad 1 \le j \le M \qquad (3.5d)$$

<u>Proof of Proposition 3.1</u>

See Luenberger [12].

Now observe that: $$\frac{\partial E(\vec{R}^*)}{\partial r_j} = \frac{de_j(r_j^*)}{dr_j} \qquad (3.6)$$

We can reduce the preceding set of conditions by eliminating the presence of both $\vec{\gamma}^*$ and $\vec{U}^*$.

This can be done as follows:

Examine: (3.5c), and (3.5d), we have three cases to consider:

(i) $0 < r_j^* < r_j^{Max}$

(ii) $r_j^* = 0$

(iii) $r_j^* = r_j^{Max}$

<u>Case (i)</u>: Assume $0 < r_j^* < r_j^{Max}$

Then (3.5c) implies $\gamma_j^* = 0$ and

(3.5d) implies $u_j^* = 0$.

Applying the preceding results to (3.5a) and multiplying by -1, we have ;

$$\frac{de_j(r_j^*)}{dr_j} - \sum_{i=1}^{M} h_{ij}\, \lambda_i^* = 0 \qquad \begin{array}{c} 0 < r_j^* < r_j^{Max} \\ 1 \le j \le M \end{array}$$

as desired.

<u>Case (ii)</u>: Assume $r_j^* = 0$

Then (3.5c) implies $\gamma_j^* \geq 0$

and (3.5d) implies $u_j^* = 0$

Applying this result to (3.5a) we have:

$$\frac{de_j(r_j^*)}{dr_j} - \sum_{i=1}^{M} \lambda_i^*\, h_{ij} = -\gamma_j^* \leq 0 \qquad \begin{array}{c} \text{for } r_j^* = 0 \\ 1 \le j \le M \end{array}$$

as desired.

Case (iii) Assume $r_j^* = r_j^{Max}$

Then (3.5c) implies $\gamma_j^* = 0$

and (3.5d) implies $u_j^* \geq 0$

Applying this result to (3.5a) we have:

$$\frac{de_j(r_j^*)}{dr_j} - \sum_{i=1}^{M} \lambda_i^* h_{ij} = u_j^* \geq 0 \qquad \text{for } r_j^* = r_j^{Max}$$

$$1 \leq j \leq M$$

Combining the three cases and noting that $\lambda_i^* \geq 0$, $1 \leq i \leq N$, we get the desired result. Q.E.D.

Having completed the formulation of the optimization problem and the statement of optimality conditions we are now ready to proceed with the development of the distributed algorithm.

3.3 Distributed Algorithms: A General Discussion.

A distributed algorithm is one in which the links cooperate in an organized fashion to perform the desired network optimization. Thus we associate with each link a control value which is computed on the basis of local information and resources. Let us denote the control value for link i as S_i.

For each user there is a relationship which allows us to determine the user's rate as a function of the link control values associated with the user's route.

Basically to solve the network optimization problem the link control values are varied in an appropriate manner until the optimum user rates have been assigned and from this point on in time the control values should remain constant. Each time the link control values are updated the appropriate information regarding each user's route must be communicated to the users. Two practical methods for accomplishing this are as follows.

Method (1): There is a special data field in the user's packet denoted control information. Each time the packet passes through a link along its route, the link processor reads the information in the control field and then using its current control value performs the data manipulation. The resultant is then written into the control field. When the packet finally arrives at its destination the information residing in the control field is somehow communicated back to the source.

Method (2): Periodically each node in the network broadcasts to all its neighbors the identity and current control value of each of its outgoing links. When a node receives such a list, it rebroadcasts this list to all its neighbors. Eventually by flooding every source in the network will learn the link control value for every link in the network. Then each source can simply compute its rate as function of the link control values, assuming it knows the path associated with its conversation.

In practice there must be a finite period of time between control value updates and hence changes in user rates. A certain period of time is necessary for each link to gather local information about its status and to perform the control update computation. We will call this period of time the link observation period, denoted d_O. In addition because of the presence of finite feedback delay a period of time is necessary to communicate the user's current control information from the destination back to the user. We will call this period of time the feedback delay, denoted d_F.

It should be clear from the preceding discussion that we are dealing with a continuous time process upon which we make control value updates and user rate changes at discrete time increments. Fig. 3.1 illustrates the basic interaction between the various time periods.

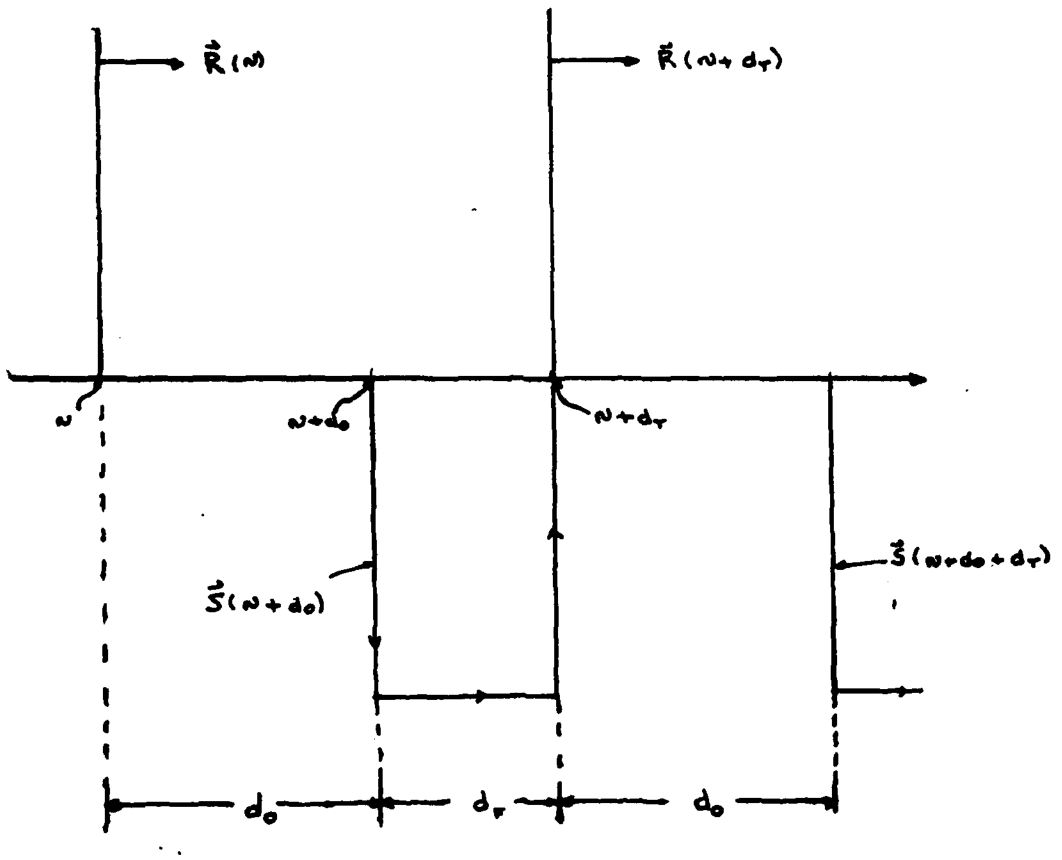


Fig. 3.1 Discretization of Control Process

It is obvious from Fig. 3.1 that the fundamental time duration between either control or rate updates is simply the sum of the observation period (d_o) and the feedback delay period (d_F). We shall denote this period of time by the variable d_T, that is $d_T = d_o + d_F$

With the preceding conclusions in mind, in parallel with our earlier notation let us now make the following definitions.

Definition 3.1 Let $r_i(k)$ denote the rate of user u_i in (bits/sec) for the time period $[k, k+d_T]$. In vector form

$$\vec{R}(k) = \begin{bmatrix} r_1(k) \\ \cdot \\ \cdot \\ \cdot \\ r_M(k) \end{bmatrix} \quad \text{(user - rate vector)}$$

Definition 3.2 Let $f_j(k)$ denote the flow of traffic on link j in (bits/sec) for the time period $[k, k+d_T)$. In vector form we have:

$$\vec{F}(k) = \begin{bmatrix} f_1(k) \\ \cdot \\ \cdot \\ \cdot \\ f_N(k) \end{bmatrix} \quad \text{(link - flow vector)}$$

Definition 3.3 Let $S_j(i)$ denote the link control value for link j for the time period $[i - d_F, i+d_o]$. In vector form we have:

$$\vec{S}(i) = \begin{bmatrix} S_1(i) \\ \cdot \\ \cdot \\ \cdot \\ S_N(i) \end{bmatrix} \quad \text{(link - control vector)}$$

Now note that we can use def. 3.1 and 3.2 in (2.1) and (2.2) to get (3.6) and (3.7) respectively.

$$f_i(k) = \sum_{j=1}^{M} h_{ij}\, r_j(k) \tag{3.6}$$

$$\vec{F}(k) = H\vec{R}(k) \tag{3.7}$$

Below we portray in an analytical format a typical sequence of algorithm operations

$$\vec{S}(k) \rightarrow \vec{R}(k) \rightarrow \vec{F}(k) \rightarrow \vec{S}(k+d_T) \rightarrow R(k+d_T) \rightarrow F(k+d_T) \tag{3.8}$$

or equivalently

$$\vec{S}(N) \rightarrow \vec{R}(N) \rightarrow \vec{F}(N) \rightarrow \vec{S}(N+1) \rightarrow \vec{R}(N+1) \rightarrow \vec{F}(N+1) \tag{3.9}$$

To get (3.9) from (3.8) we simply normalize our fundamental time basis by d_T. Unless otherwise specified we will assume for now on that d_T has been normalized to 1.

3.4 Development of Distributed Flow Control Algorithm

The objective of this section is to develop an algorithm which solves for the optimum user rates of (3.3) in an iterative way using distributed computation. In order to gain some insight into how this may be accomplished it is first useful to view λ_i as the "cost" of link i. Basically it can be interpreted as the incremental "cost" of sending flow on link i. With this idea in mind we can view $\sum_{i=1}^{M} h_{ij}\, \lambda_i$ as the cost of user u_j's route, since it is simply the sum of the costs of all links contained in u_j's route.

It is clear that we should choose the link control values $\{S_i\}$ to be completely equivalent with $\{\lambda_i\}$and the two sets are distinguished only for notational convenience.

In the preceding section we found that in order to develop a distributed algorithm two basic relationships must be determined. They are:

(i) the relationship between the rate of each user and the link control values corresponding to this route

(ii) The link control value update equation.

We use equation (3.4a) to specify (i) as

$$\frac{de_j(r_j)}{dr_j} = \sum_{i=1}^{N} h_{ij}\,\lambda_i \qquad 1 \le j \le M \qquad (3.10)$$

At this point we make the following definitions:

<u>**Definition 3.5**</u> The function $g_j(r_j)$ of user u_j, $1 \le j \le M$ is its marginal reward function, i.e.

$$g_j(r_j) \triangleq \frac{de_j(r_j)}{dr_j} \qquad (3.11)$$

The function $g_j(r_j)$ has the interpretation that it is the incremental gain for additional allocation to user u_j.

Now using (3.11) in (3.10)we get

$$g_j(r_j) = \sum_{i=1}^{N} h_{ij}\,S_i \qquad 1 \le j \le M \qquad (3.12)$$

Definition 3.7 Let d_j represent the total cost of user u_j's route.

Thus:

$$d_j = \sum_{i=1}^{M} h_{ij} S_i \qquad 1 \leq j \leq M \qquad (3.13)$$

and

$$d_j^{Max} = \left.\frac{de_j(r_j)}{dr_j}\right|_{r_j=0} \qquad (3.13a)$$

$$d_j^{Min} = \left.\frac{de_j(r_j)}{dr_j}\right|_{r_j = r_j^{Max}} \qquad (3.13b)$$

Thus using def. 3.7 in (3.12) we have:

$$g_j(r_j) = d_j \qquad 1 \leq j \leq M \qquad (3.14)$$

In words, (3.14) states the reward for user u_j is equal to the cost of its route.

A typical $g_j(r_j)$ is shown in Figure 3.2.

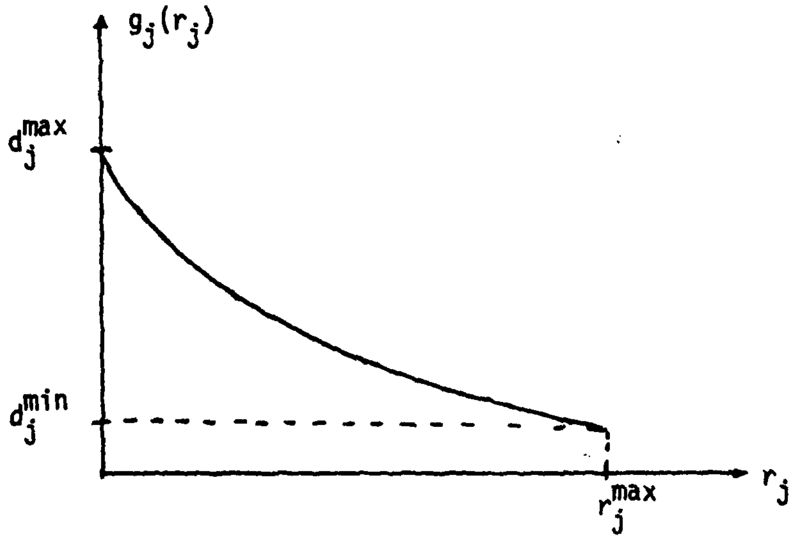


Fig. 3.2 Typical User Marginal Reward Function

Definition 3.8 Let $b_j(d_j)$, $d_j \in [0,\infty]$ be defined as follows

$$r_j = b_j(d_j) = \text{inverse of } [g_j(r_j)] \qquad d_j^{Min} \le d_j \le d_j^{Max} \tag{3.15a}$$

$$r_j = b_j(d_j) = 0 \qquad d_j \ge d_j^{Max} \tag{3.15b}$$

$$r_j = b_j(d_j) = r_j^{Max} \qquad 0 < d_j < d_j^{Min} \tag{3.15c}$$

A typical $b_j(d_j)$ is shown in Figure 3.3

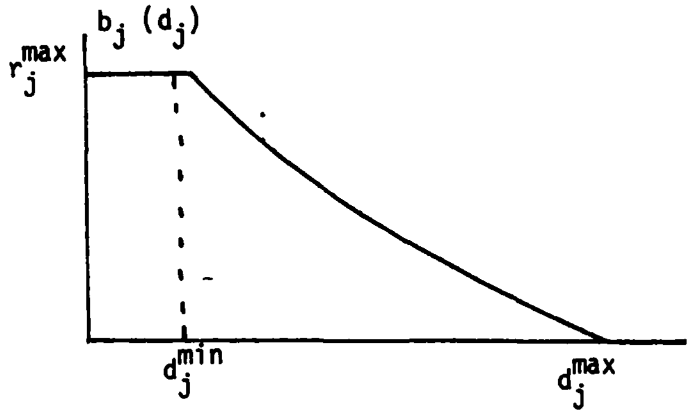


Fig. 3.3 Typical User Rate Assignment Function

Now note in vector form (3.13) becomes

$$\vec{D} = H^T \vec{S} \tag{3.17}$$

and using (3.15)

$$\vec{R} = \vec{B}(\vec{D}) = \vec{B}(H^T \vec{S}) \tag{3.18}$$

where

$$\vec{B}(\vec{D}) \triangleq \begin{bmatrix} b_1(d_1) \\ \\ b_M(d_M) \end{bmatrix} \tag{3.19}$$

Next we must specify (ii). Equation (3.4b) is the key to determining the link control update equation. It is repeated below for the reader's convenience, except we have replaced λ_i^* by the equivalent variable S_i^*

$$S_i^*[f_i^* - c_i] = 0, \qquad 1 \leq i \leq N \qquad (3.20)$$

In words (3.20) simply states that if at the optimum point link i is not saturated, then the solution value for link i's control variable, namely S_i^* will be identically equal to zero. Thus intuitively an update equation for S_i would be one which decreases its value at each iteration by a fixed amount as long as the link has not become saturated. Since S_i must be non-negative we must require that S_i have a minimum value of zero. The following definition formally defines the update equation.

Definition 3.8 Let each link j, $1 \leq j \leq N$ have associated with it the following link control value update equation.

$$S_j(\text{new}) = [S_j(\text{old}) + \epsilon_j(f_j(\text{old}) - c_j]^+ \qquad (3.21)$$

where ϵ_j is a positive constant and the notation $[x]^+ = \text{Max}(0,x)$.

In vector form we have;

$$\vec{S}(\text{new}) = [\vec{S}(\text{old}) + \varepsilon(\vec{F}(\text{old}) - \vec{C})]^{+} \tag{3.22}$$

where

$$\epsilon \triangleq \begin{bmatrix} \epsilon_1 & & \bigcirc \\ & \epsilon_2 & \\ \bigcirc & & \ddots \; \epsilon_N \end{bmatrix} \quad \text{with } \varepsilon_i > 0$$

and the notation $[\vec{x}]^{+} = \begin{bmatrix} (x_1)^{+} \\ (x_N)^{+} \end{bmatrix}$

Now using (3.7) in (3.22) we get:

$$\vec{S}(\text{new}) = [\vec{S}(\text{old}) + \underset{\sim}{\epsilon} (H \vec{R}(\text{old}) - \vec{C}\,]^{+} \tag{3.23}$$

In order to get a mapping from $\vec{S}(\text{old}) \to \vec{S}(\text{new})$, we need to express $\vec{R}(\text{old})$ in terms of $\vec{S}(\text{old})$. This can be accomplished through the use of (3.18). Thus we finally arrive at the desired mapping:

$$\vec{S}(\text{new}) = [\vec{S}(\text{old}) + \varepsilon\, [H\, \vec{B}\, (\vec{H}^{T}\, \vec{S}(\text{old}) - \vec{C}\,]]^{+} \tag{3.25}$$

Let us denote this mapping by Z. Hence we have

$$\vec{S}(\text{old}) = Z\,[\vec{S}(\text{old})] = \vec{S}(\text{new}) \tag{3.26}$$

where $Z : \mathbb{R}^{N} \to \mathbb{R}^{N}$.

3.5 Proof of Convergence

Theorem 3.2 Let $e_j(r_j)$ be a continuous convex function for $r_j \in [0, r_j^{Max}]$. The second derivative, i.e. $e_j''(r_j)$ is piecewise continuous and smaller than $(-1/k_j)$.

If the ε matrix is appropriately chosen, then the algorithm described by (3.15) and (3.23) produces a sequence $\vec{R}(N)$ converging to a $\vec{R}^*$ maximizing $E(\vec{R})$ subject to $H\vec{R} \leq \vec{C}$ and $0 \leq \vec{R} \leq \vec{R}^{Max}$.

Proof of Theorem 3.2

The function $e_j(r_j)$ is strictly convex and as a result the objective function $E(\vec{R})$ has a unique maximum. Furthermore, the derivative of $e_j(r_j)$ is strictly decreasing. Thus the function $b_j(d_j)$ defined in (3.15) is well defined, continuous and piecewise differentiable.

As the e_i are convex $\cap$ and differentiable and the allowable region for $\vec{R}$ is convex, it is well known (e.g. See Luenberger [12]) that:

$$\max_{\substack{\vec{R}\geq 0\\ H\vec{R}\leq\vec{C}\\ \vec{R}\leq\vec{R}^{Max}}} E(\vec{R}) = \max_{\substack{\vec{R}\geq 0\\ \vec{R}\leq\vec{R}^{Max}}} \min_{\vec{S}\geq 0} [E(\vec{R}) + \vec{S}^T(\vec{C} - H\vec{R})] \tag{3.27}$$

or equally,

$$\max_{\substack{\vec{R}\geq 0\\ H\vec{R}\leq\vec{C}\\ \vec{R}\leq\vec{R}^{Max}}} E(\vec{R}) = \min_{\vec{S}\geq 0} \max_{\substack{\vec{R}\geq 0\\ \vec{R}\leq\vec{R}^{Max}}} [E(\vec{R}) + \vec{S}^T(\vec{C} - H\vec{R})] \tag{3.28}$$

We denote $\underset{\substack{\vec{R}\geq 0 \\ \vec{R}\leq\vec{R}^{Max}}}{\text{Max}} [E(\vec{R}) + \vec{S}^T(\vec{C} - H\vec{R})]$ by $\phi(\vec{S})$ (3.29)

Then,

$$\underset{\substack{\vec{R}\geq 0 \\ H\vec{R}\leq\vec{C} \\ \vec{R}\leq\vec{R}^{Max}}}{\text{Max}} E(\vec{R}) = \underset{\vec{S}\geq 0}{\text{Min}} \phi(\vec{S}) \tag{3.30}$$

Note that for a given $\vec{S}$, $[E(\vec{R}) + \vec{S}(\vec{C} - H\vec{R})]$ is maximized by an $\vec{R}$ which satisfies:

$$(\nabla_{\vec{R}} E(\vec{R}) - \vec{S}^T H)_j \begin{cases} = 0, & 0 \leq r_j \leq r_j^{Max} \quad (3.31a) \\ \leq 0, & r_j = 0 \quad (3.31b) \\ \geq 0, & r_j > r_j^{Max} \quad (3.31c) \end{cases}$$

In other words the optimal $\vec{R}$ is $\vec{B}(H^T \vec{S})$. This is effectively implemented in (3.15).

Where it exists, the gradient of $\phi(\vec{S})$ is given by:

$$\nabla_{\vec{S}} \phi(\vec{S}) = \nabla_{\vec{R}} [E(\vec{R}) + \vec{S}^T(\vec{C} - H\vec{R})]\Big|_{\vec{R}=\vec{B}(H^T\vec{S})} \nabla_x \vec{B}(\vec{x})\Big|_{\vec{x}=H^T\vec{S}} H^T + (\vec{C} - H\vec{R})^T\Big|_{\vec{R}=\vec{B}(H^T\vec{S})} \tag{3.32}$$

Note in (3.32) that the jth component of the first factor on the right hand side is 0 if $0 < r_j < r_j^{Max}$, whereas the jth component of the second factor is 0 for $r_j < 0$ or $r_j > r_j^{Max}$. Thus the product of the first two factors is identically 0 (the identity on all boundaries is checked by examining the left and right derivatives). Hence:

$$\nabla_{\vec{S}} \phi(\vec{S}) = (\vec{C} - H\vec{R})^T. \tag{3.33}$$

Thus (3.23) is just a steepest descent algorithm for $\phi(\vec{S})$.

The matrix of partial second derivatives of $\phi(\vec{S})$ is:

$$-H\nabla_{\vec{S}} R(H^T \vec{S}) = -H[e''_j (r_j)]^{-1} H^T \le H K H^T \tag{3.34}$$

where

$$[e''_j (r_j)] = \begin{bmatrix} e''_1(r_1) & & & & \\ & \ddots & & \bigcirc & \\ & & e''_j(r_j) & & \\ & \bigcirc & & \ddots & \\ & & & & e''_M(r_j) \end{bmatrix} \tag{3.35}$$

and

$$K = \begin{bmatrix} k_1 & & & & \\ & \ddots & & \bigcirc & \\ & & k_j & & \\ & \bigcirc & & \ddots & \\ & & & & k_M \end{bmatrix} \tag{3.36}$$

Now denote by $\vec{\Delta}^N$ the vector such that:

$$\vec{S}(N+1) = \vec{S}(N) - \varepsilon \vec{\Delta}^N \tag{3.37}$$

Note in (3.37) that Δ_i^N is either equal to $(\vec{C} - H\,\vec{R}(N))_i$ or (if $s_i(N+1)$ is 0) $0 \le \Delta_i^N \le (\vec{C} - H\,\vec{R}(N))_i$.

Hence,

$$[\vec{C} - H\,\vec{R}(N)]^T \varepsilon\vec{\Delta}^N \ge (\vec{\Delta}^N)^T \varepsilon\vec{\Delta}^N \tag{3.38}$$

Now by the mean value theorem,

$$\phi(\vec{S}(N+1) \le \phi(\vec{S}(N)) - (\vec{C} - H\vec{R}(N))^T \varepsilon\vec{\Delta}^N + (\tfrac{1}{2}) (\vec{\Delta}^N)^T \varepsilon HK \varepsilon\Delta^N \tag{3.40}$$

We must now find the conditions on ε such that if $\vec{\Delta}^N \ne \vec{0}$, then $\phi(\vec{S}(N+1)) < \phi(\vec{S}(N))$.

Thus we desire for $\vec{\Delta}^N \neq \vec{0}$,

$$(\vec{\Delta}^N)^T \quad \varepsilon(\vec{\Delta}^N) > \frac{1}{2}(\vec{\Delta}^N)^T \varepsilon HK\ H^T\ \varepsilon(\vec{\Delta}^N) \tag{3.41}$$

Since ε is diagonal, $\varepsilon = (\varepsilon^{1/2})^T \quad (\varepsilon^{1/2})$

$$\text{where: } \varepsilon^{1/2} = \begin{bmatrix} \varepsilon^{1/2} & & & \\ & \cdot & & \\ & & \cdot & \\ & & & \varepsilon_N^{1/2} \end{bmatrix} \tag{3.42}$$

So (3.41) becomes

$$(\vec{\Delta}^N)^T \ (\varepsilon^{1/2})^T \ (\varepsilon^{1/2})\vec{\Delta}^N > \frac{1}{2} (\vec{\Delta}^N)^T \ (\varepsilon^{1/2})[(\varepsilon^{1/2}) \ HKH^T \ (\varepsilon^{1/2})^T](\varepsilon^{1/2})\vec{\Delta}^N \tag{3.43}$$

Let $\vec{X} = (\varepsilon^{1/2}\ \vec{\Delta}^N)$, then (3.43) becomes

$$\vec{X}\,[I - \frac{1}{2}(\varepsilon^{1/2}\ HKH^T\ \varepsilon^{1/2})]\ \vec{X} > 0 \qquad \forall\ \vec{X} \neq \vec{0} \tag{3.44}$$

$$\text{Let } Q \stackrel{\Delta}{=} [I - (\frac{1}{2}\ \varepsilon^{1/2}\ HK\ H^T\ \varepsilon^{1/2}] \tag{3.45}$$

Then (3.44) becomes

$$\vec{X}^T Q\ \vec{X} > 0 \qquad \forall\ \vec{X} \neq \vec{0} \tag{3.46}$$

Since Q is a real symmetric matrix (3.45) is just the statement that Q is positive definite. Now a necessary and sufficient condition on Q to be positive definite is that all its eigenvalues be strictly greater than zero. This implies (See Strang [13]),

$$\boxed{\text{Maximum eigenvalue of } (\varepsilon^{1/2}\ HK\ H^T\ \varepsilon^{1/2}) < 2} \tag{3.47}$$

Thus assuming (3.47) is valid, as long as $\vec{\Delta}^N \neq 0$, $\phi(\vec{S}(N+1)) < \phi(\vec{S}(N))$. As the mappings (3.15), (3.23) and $(\vec{S})$ are continuous and since $\vec{S}$ is bounded, the global convergence theorem in [12] guarantees that any convergent subsequence of the sequence $\vec{S}(N)$ converges to a point minimizing $\phi(\vec{S})$ on $\vec{S} \geq 0$. By strict convexity this point is unique and the sequence $\vec{S}(N)$ converges to it. Thus (3.4) is satisfied and hence convergence of the algorithm is guaranteed. QED.

3.6 Comments on the Distributed Algorithm

The algorithm presented in this chapter is a reasonable and easily implementable flow control algorithm. However, it has two major drawbacks which are discussed below.

(1) Convergence of the algorithm is only guaranteed if condition (3.47) holds. Let us temporarily assume K = kI, then (3.50) becomes:

$$\text{Maximum eigenvalue } (\varepsilon^{1/2} HH^T \varepsilon^{1/2}) < (2/k) \qquad (3.48)$$

Now

$$HH^T \overset{\Delta}{=} W = \begin{bmatrix} W_{11} & & W_{1N} \\ & W_{ij} & \\ W_{N1} & & W_{NN} \end{bmatrix} \qquad (3.49)$$

where, $W_{ij} = \sum_{k=1}^{M} h_{ij} h_{jk}$.

Hence W is a real symmetric, where W_{ij} represents the number of users who utilize both link i and link j.

It is well known (see Strang [13]) that the maximum eigenvalue of a matrix is always less than the maximum row sum. Using this fact and (3.49) condition (3.48) becomes:

$$\underset{i}{\text{Max}} \; \varepsilon_i^{1/2} \left[\sum_{j=1}^{N} \varepsilon_j^{1/2} \; W_{ij} \right] < (2/k) \qquad (3.50)$$

Now assuming $\varepsilon_j < (2/k) \quad 1 \le j \le N$, (3.50) is satisfied if

$$\varepsilon_i < \frac{(2/k)}{\left(\sum_{j=1}^{N} W_{ij}\right)^2} < \frac{2}{k} \tag{3.51}$$

Thus (3.51) states that each link i must somehow determine the quantity: $\sum_{j=1}^{N} W_{ij}$. Hence each user who utilizes link i must inform link i of the total number of links which it utilizes in its route. This would be done at time of call set-up.

If we assume $\varepsilon_j < (2/k\, W_{ij})$, $1 \le j \le N$, (3.50) is satisfied

if
$$\varepsilon_i < \frac{2}{k\left(\sum_{j=1}^{N} \sqrt{W_{ij}}\right)^2} < \frac{2}{k\, W_{ij}} \tag{3.52}$$

In this case each user who utilizes link i must inform link i of all the other links which it utilizes in its route.

However with restrictions (3.51) or (3.52) the algorithm is no longer completely distributed since each link must know a certain amount of information concerning the utilization of the other links in the network.

(2) The major drawback of the algorithm is that it does not treat all network users in the same manner, even if they have the same e_i's. This can be observed by noting that the rate of a particular user

say u_j, is a function of the quantity: $\sum_{k \epsilon L_j} S_k$, namely the cost of u_j's route. Clearly the more links in user u_j's route the more likely it is to have a higher assigned cost, and thus by Fig. 3.2 a lower assigned rate. Hence, the algorithm penalizes those users who require many links to construct their route, which contradicts the standard policy of voice communication networks (e.g. the common carrier telephone network).

As a result of the preceding two drawbacks we terminated work on this algorithm in an effort to develop an algorithm which would be completely distributed and completely "fair" to all network users.

3.7 Example

We conclude this chapter with a simple example. Consider the network in Fig. 3.4

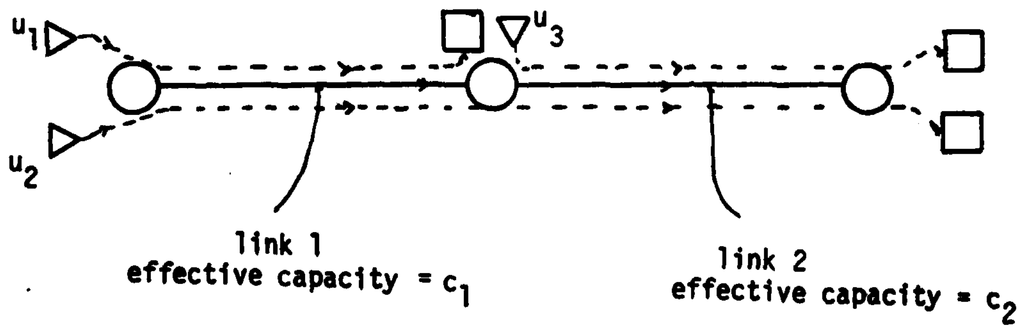


Figure 3.4 Simple 3 User, 2 Link Network

Let $e_j(r_j) \triangleq -(r_j - r_j^{Max})^2 \qquad j = 1,2,3$

and $r_j^{Max} = C \qquad j = 1,2,3$

Then $\dfrac{de_j(r_j)}{dr_j} = -2(r_j - c)$

By (3.11)

$$g_j(r_j) = -2(r_j - c)$$

and by (3.13)

$$d_j^{Max} = 2c, \quad d_j^{Min} = 0$$

Hence (3.14) and (3.15) gives us

$$r_j = \left[\frac{2c - d_j}{2}\right] \qquad \text{for} \quad 0 \leq d_j \leq 2c$$

$$r_j = 0 \qquad \text{for} \quad d_j > 2c$$

Using (3.12) we have

$$\begin{aligned} d_1^* &= S_1^* + S_2^* \\ d_2^* &= S_1^* \\ d_3^* &= S_2^* \end{aligned}$$

The optimal solution of (3.4) is

$$\vec{S}^* = \begin{bmatrix} 2c/3 \\ 2c/3 \end{bmatrix}$$

So by (3.17)

$$r_1^* = c/3$$
$$r_2^* = 2c/3$$
$$r_3^* = 2c/3$$

The proceeding example clearly shows that the rate assignment for a particular user depends upon the number of links which the user utilizes.

CHAPTER IV

THE FAIR FLOW CONTROL SCHEME

We consider in this chapter a flow control scheme which has user "fairness" as its primary objective.

After motivating the scheme and presenting the centralized algorithm, we demonstrate its equivalence to a sequential optimization problem. The distributed algorithm is then developed and shown to guarantee convergence to the unique user rate assignment given by the centralized algorithm. The chapter concludes by examining extensions of the original algorithm.

4.1 Introduction

In this chapter we develop a flow control scheme which is based upon a reasonable notion of fairness. By fairness we mean that the quality of service that each user receives is dependent only upon the current network traffic conditions and independent of the actual length of the user's route (measured by physical distance or by the number of links used). Furthermore in order to be fair in an economic sense we must assume the amount each user pays for network service is proportional to the amount of network resources it utilizes. A familiar example which illustrates this viewpoint is the common carrier telephone network.

At first thought, one can easily insure fairness by assigning each user the same rate. However, in a network with different users, utilizing links of different capacities, it is improbable that such a scheme would be desirable. A second approach is to somehow equally divide the network resources among the users and it is this approach which we will follow in our development.

4.2 Development of the Scheme

To construct a flow control scheme we must determine the policy which governs the allocation of network resources. In selecting appropriate user rates two basic requirements must be satisfied:

(i) the steady-state total flow on each link must not exceed the effective link capacity

(ii) the quality of each user's service must be as high as possible.

The following simple examples serve to motivate the development of our FAIR RATE assignment scheme.

Example 4.1

Consider the simple network illustrated in Fig. 4.1

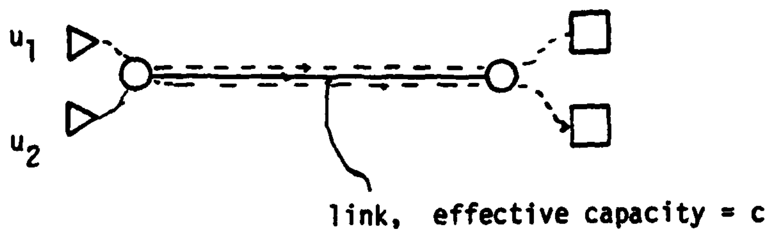


Figure 4.1 Simple two User, one Link Network

Since our primary objective is fairness, any rate assignment other than $r_1^* = r_2^* = (c/2)$, would be unfair to either user u_1 or user u_2 because it would imply either $r_1^* < r_2^*$ or $r_1^* > r_2^*$.

Example 4.2

Consider the network shown in Fig. 4.2

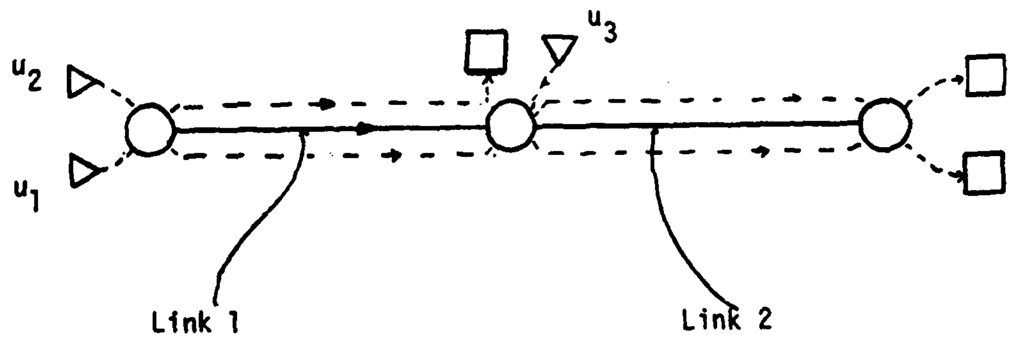


Figure 4.2 Three User, two Link Network

Let's examine a few cases of this network.

Case 1) Let $c_1 = c_2 = c$

Then the fair rate assignment is simply given by $r_1^* = r_2^* = r_3^* = (c/2)$. This particular assignment also happens to achieve full utilization of the network resources (all links are saturated, i.e. $f_i^* = c_i$, $i = 1,2$).

Case 2) Let $c_1 = c/2$, $c_2 = c$

In this case one must be careful of the order in which user rates are assigned. For instance, suppose we first divide up the resources of link 2 equally among its two users (i.e. $r_1^* = r_3^* = (c/2)$). Then the residual capacity of link 1 is equal to zero, hence since requirement (i) must be satisfied this implies the rate assignment for user 2, i.e. r_2^* is equal to zero.

Thus even though we have achieved full utilization of the network resources we have arrived at an <u>unfair</u> solution, since user u_2 is blocked. The fair rate assignment for this case would be determined by first dividing up the resources of link 1 equally among its two users, $r_1^* = r_2^* = (c_1/2) = (c_{/4})$, then assigning $r_3^* = c_2 - r_1^* = (3c/4)$.

It is very important to note that the fair rate assignment does not imply that all links in the network will be saturated. To illustrate this point consider the following example.

Example 4.3

Consider the network illustrated in Fig. 4.3

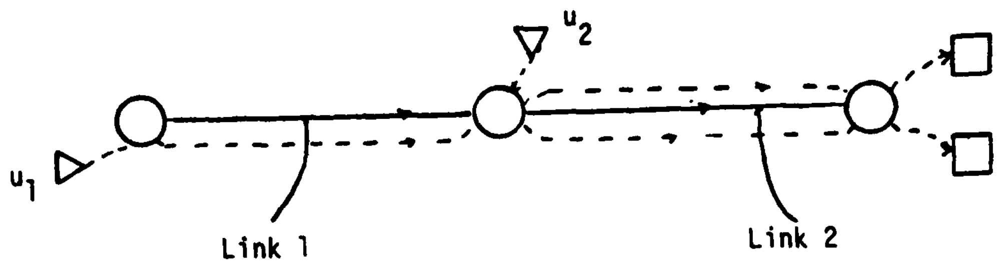


Figure 4.3 Two User, two Link Network

Let $c_1 = c_2 = c$.

Then the fair rate assignment is simply $r_1^* = r_2^* = (c/2)$. Now note that the steady state flow on link 1, denoted $f_1^* = r_1^* = (c/2)$, and hence link 1 is not saturated. However, in general if each link i has at least one user which utilizes only link i, then the fair rate assignment always results in full utilization of the network resources.

At this point we need to develop an algorithmic approach for solving the fair rate assignment problem for general networks. Thus we formalize the fair rate assignment algorithm as follows.

Definition 4.1 Let $\mathcal{U}_j$ be the set of users who utilize link j, i.e.

$$\mathcal{U}_j = \{u_i | h_{ji} = 1\}$$

$$\text{Note } \mathcal{U} = \bigcup_{j=1}^{M} \mathcal{U}_j \qquad (4.1)$$

where, U denotes union.

__Definition 4.2__ Let L_k be the set of all links used by user u_k, i.e.

$$L_k = \{j | h_{jk} = 1\}$$

__Definition 4.3__ Let W_j = the number of users who utilize link j.

__Definition 4.4__ Let w_{jk} = the number of users who utilize both link j and link k.

Then the Fair Flow control algorithm can be stated as follows:

FAIR FLOW CONTROL ALGORITHM (A)

__Step 1:__ Determine the link(s) which is(are) currently the network bottleneck(s). This is done by finding all links j^* s.t.

$$\left(\frac{c_{j*}}{W_{j*}}\right) \leq \left(\frac{c_j}{W_j}\right) \qquad \forall\ j \neq j^* \qquad (4.3)$$

Denote the set of all such links by J^*, and the value $\left(\frac{c_{j*}}{W_{j*}}\right)$ by γ .

__Step 2:__ Assign all users who utilize a

link $j^* \varepsilon J^*$ the rate γ ,

$$\text{i.e. } r_k^* = \gamma \quad \forall\ k \varepsilon L_{j*} \ , \quad \forall\ j^* \varepsilon J^* \qquad (4.4)$$

__Step 3:__ Reduce the original problem by eliminating the presence of all users assigned in step (2).

Thus let

$$\mathcal{U} = \{\mathcal{U} - \bigcup_{j^*\varepsilon J^*} \mathcal{U}_j^*\} \tag{4.5}$$

$$L = \{L - \bigcup_{j^*\varepsilon J^*} L_j^*\} \tag{4.6}$$

$$c_j = \{c_j - \sum_{j^*\varepsilon J^*} w_{jj^*}\gamma\} \qquad \forall_j \varepsilon L \tag{4.7}$$

$$W_j = \{W_j - \sum_{j^*\varepsilon J^*} w_{jj^*}\} \qquad \forall_j \varepsilon L \tag{4.8}$$

<u>Step 4.</u> Repeat steps (1),(2) and (3) until all users have been assigned a rate, i.e. until $\mathcal{U} = \{\emptyset\}$.

Some important properties of this algorithm are as follows:

(1) We consider the algorithm to be fair because the rate of each user is greater than or equal to the rate of all users that share its bottleneck link.

(2) By the way we assign rates we are guaranteed that each link will have a steady-state flow which does not exceed the effective link capacity of the link.

(3) The rate assignment is unique.

(4) At each iteration of the procedure we are essentially maximizing the minimum user rate by equally dividing up the resources of the current bottleneck link(s) among those users on that (those) link(s) (i.e. the set J^*) that have not already been assigned a rate.

As a result of property (4) rates are always assigned in order of increasing magnitude.

Formally this can be shown as follows. We have from Step 1,

$$\left(\frac{c_j}{W_j}\right) \geq \left(\frac{c_{j*}}{W_{j*}}\right) = \gamma \qquad \forall_j \notin J^* \tag{4.9a}$$

and from Step 3,

$$\hat{c}_j = c_j - k_j \gamma \qquad \forall_j \notin J^* \tag{4.9b}$$

$$\hat{w}_j = W_j - k_j \qquad \forall_j \notin J^* \tag{4.9c}$$

where

$$k_j = \sum_{j^* \varepsilon J^*} w_{j^*} \qquad \forall_j \notin J^* \tag{4.9d}$$

then, using (4.9a) in (4.9b)

$$\frac{\hat{c}_j}{\hat{W}_j} = \frac{c_j - k_j\gamma}{w_j - k_j} > \frac{\gamma\,(w_j - k_j)}{w_j - k_j} = \gamma \qquad \forall_j \notin J^* \tag{4.9e}$$

which is the desired property.

The latter property of the algorithm allows us formulate the fair rate assignment problem as the following iterative optimization problem.

<u>FAIR FLOW CONTROL ALGORITHM (B)</u>

<u>Step 1</u> Max z (4.10a)

S.t. $r_j \geq z \qquad \forall\, j\, \varepsilon\, \mho$ (4.10b)

$\sum_{j \varepsilon \mho} h_{ij}\, r_j \leq c_i \qquad \forall\, i\, \varepsilon\, L$ (4.10c)

$r_j \geq 0 \qquad \forall\, j\, \varepsilon\, \mho$ (4.10d)

The solution to this problem is a set of user rates denoted $\{r_j^*\}$ and resultant link flows $\{f_i^*\}$ where,

$$f_i^* = \sum_{j \in \mathcal{U}} h_{ij} r_j^* \qquad (4.11)$$

<u>Step 2</u> Let $\mathcal{U}^* = \{u_j | r_j^* = z\}$ (4.12)

Assign all users $u_j \in \mathcal{U}^*$ the rate z.

<u>Step 3</u> Reduce the original problem as follows

Let: $L^* = \{j | f_j^* = c_j\}$ (4.13)

then:

$$\mathcal{U} = \{\mathcal{U} - \mathcal{U}^*\}$$

$$L = \{L - L^*\} \qquad (4.15)$$

$$c_j = c_j - \sum_{k \in L^*} w_{jk} z \quad \forall\ j \in L \qquad (4.16)$$

$$W_j = W_j - \sum_{k \in L^*} w_{jk} \quad \forall\ j \in L \qquad (4.17)$$

<u>Step 4</u> Repeat steps (1), (2) and (3) until all users have been assigned a rate, i.e. until $\mathcal{U} = \{\emptyset\}$.

Let us now present one final example to illustrate the application of the general fair flow control algorithm.

<u>Example 4.4</u> Consider the network shown in Fig. 4.4.

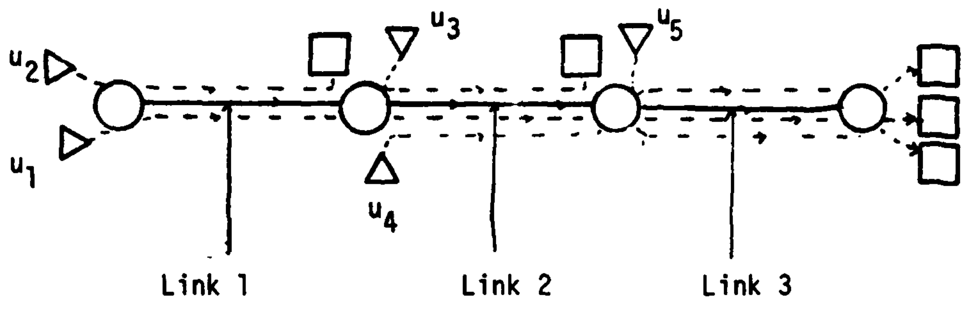


Figure 4.4 Five User, Three Link Network

Let $c_1 = 2c$, $c_2 = c$, $c_3 = c/2$

Then the fair rate assignment proceeds as follows:

Step 1A $\quad \frac{c_3}{w_3} = \frac{c}{3} < \frac{c_i}{w_i} \quad i = 1,2$

so $j^* = 3 \quad J^* = \{3\}$

Step 2A $\quad r_1^* = r_4^* = r_5^* = \frac{c_3}{w_3} = \frac{c}{6} = \gamma$

Step 3A $\quad \mathcal{U} = \{u_2, u_3,\}$

$L = \{1,2\}$

$c_1 = c_1 - \gamma = 2c - c/6 = \frac{11c}{6}$

$c_2 = c_2 - 2\gamma = c - c/3 = \frac{2}{3}c$

$w_1 = w_1 - 1 = 1$

$w_2 = w_2 - 2 = 1$

Step 1B
$$\frac{c_2}{w_2} = \frac{8c}{12} < \frac{c_1}{w_1} = \frac{11c}{6}$$

$$J^* = \{2\}$$

Step 2B
$$r_3^* = \frac{c_2}{w_2} = \frac{2c}{3} = \gamma$$

Step 3B
$$\mathcal{U} = \{u_2\}$$

$$L = \{1\}$$

$$c_1 = c_1 - \gamma = \frac{11c}{6} - \frac{4c}{6} = \frac{7c}{6}$$

$$w_1 = w_1 - 1 = 1$$

Step 4B
$$r_2^* = \frac{c_1}{w_1} = \frac{7c}{6}$$

All users have been assigned rates.

The flow control algorithm that has been developed in this section is classified as a centralized algorithm since all information about the network and users' requirements must be collected in a central facility which first carries out the algorithm and then broadcasts the results to all users. Thus, we will call this algorithm the Centralized Fair Flow control algorithm.

4.3 Distributed Algorithm

In developing a distributed (decentralized) version of the Fair Rate assignment algorithm, we will follow our discussion of distributed algorithms presented in Section 3.3. Thus we associate with each network link i a control value which we denote by p_i. To determine the distributed algorithm we must specify the control value update equations and the user rate assignment relationships.

Two observations regarding the Fair Rate assignment algorithm given in the preceding section provide us with valuable insight into how the desired relations may be derived. They are as follows:

(I) the steady-state rate of each user is determined by its bottleneck link only,

(II) the steady-state rates are always assigned in order of increasing magnitude.

With the preceding observations in mind we let p_i represent the maximum rate in (bits/sec) at which link i allows all its users to transmit. Since each user may utilize several links, we set the rate of each user to the minimum control value over all links in its route. More formally, we have:

$$r_k = \min_{j \in L_k} p_j \qquad 1 \leq k \leq M \tag{4.18}$$

Next we must specify the link control value update equations. The individual objective of each link is to determine the rate for its users such that the link may become saturated. Each link makes control decisions based upon local information only, hence it must always "assume" that it has control over all its users (i.e. that changing its p will affect its users' rates), whereas in actuality it may not. So basically p_i (old) is chosen with the assumption by link i that $p_i(\text{old}) \rightarrow f_i(\text{old})$ s.t. $f_i(\text{old}) = c_i$. If indeed $f_i(\text{old}) = c_i$ then link i is satisfied and it sets $p_i(\text{new}) = p_i(\text{old})$.

However if $f_i(\text{old}) \neq c_i$, then link i must assume it has made an error in the selection of $p_i(\text{old})$ and distribute this error equally among its users. This leads us to the desired relationship defined as follows:

<u>Definition 4.5</u> Associated with each link i there is a link control value update equation defined as follows:

$$p_i(\text{new}) = p_i(\text{old}) + \frac{1}{w_i}[c_i - f_i(\text{old})] \qquad 1 \leq i \leq N \qquad (4.19)$$

To get a mapping from $p_i(\text{old}) \rightarrow p_i(\text{new})$ we must express $f_i(\text{old})$ in terms of $p_i(\text{old})$, which can be done in two steps as follows,

(i) From (3.11) we have

$$f_i(\text{old}) = \sum_{j=1}^{M} h_{ij}\, r_j(\text{old}) \qquad (4.20)$$

(ii) From (4.18) we have

$$r_j(\text{old}) = \min_{k \in L_j} p_k(\text{old}) \qquad (4.21)$$

So substituting (4.21) into (4.20) and the resultant into (4.19) we obtain the desired mapping

$$p_i(\text{new}) = p_i(\text{old}) + \frac{1}{w_i}[c_i - \sum_{j=1}^{M} h_{ij}\,[\min_{k \in L_i} p_k(\text{old})]] \qquad (4.22)$$

To examine how the distributed algorithm operates let us consider applying it to example 4.2 case (2).

Let $p_i(0) = c/8$, $p_2(0) = c/4$, (can be arbitrarily chosen)

from 4.21 we have

$$r_2(N) = p_1(N)$$

$$r_1(N) = \text{Min}\,[p_1(N),\ p_2(N)]$$

$$r_3(N) = p_2(N)$$

Now using (4.19) and (4.21)

Increment (N)	$p_1(N)$	$p_2(N)$	$r_1(N)$	$r_2(N)$	$r_3(N)$
0	c/8	c/4	c/8	c/8	c/4
1	(c/4)*	$\frac{9c}{16}$	(c/4)*	(c/4)*	$\frac{9c}{16}$
2	(c/4)*	$\frac{2c}{32}$	(c/4)*	(c/4)*	$\frac{21c}{32}$
3	(c/4)*	$\frac{45c}{64}$	(c/4)*	(c/4)*	$\frac{45c}{64}$
4	(c/4)*	$\frac{93c}{128}$	(c/4)*	(c/4)*	$(\frac{93c}{128})$

(*) desired steady-state value achieved

Note that $\lim\limits_{N\to\infty} p_2(N) \to \frac{3c}{4}$, hence

$$r_3^* = \lim_{N\to\infty} r_3(N) = \frac{3c}{4}$$

This example illustrates an important property of the algorithm. Note that those users which have the lowest steady-state rate (i.e. u_1, u_3)

converge to that rate in a finite number of iterations. However, the rate of user u_3, i.e. r_3 converges to its appropriate steady-state rate only in the limit sense. The reason being that link 2 never learns that the rate of user u_2 is being controlled by link 1. To make this point clear consider the following.

From the preceding example we have:

$$r_2(1) = r_2^* = c/4$$

$$p_2(2) = p_2(1) + \frac{1}{w_2}\{c_2 - f_2(1)\}$$

$$= (9/16)c + \frac{1}{w_2}\{c - (\frac{c}{4} + \frac{9}{16}c)\}$$

$$p_2(2) = \frac{c}{16}[9 + \frac{3}{w_2}]$$

In our present formulation w_2 is a fixed constant, however, let us temporarily assume that it could be a variable, denoted $\tilde{w}_2$. Furthermore let us assume that link 2 was somewhat informed that the rate of user u_2 was fixed by link 1, and thus link 2 knew it had control only over one user, namely u_3. Then link 2 could set $\tilde{w}_2 = 1$ and hence $\tilde{p}_2(2) = \frac{c}{16}[9 + 3] = \frac{3c}{4}$ which implies $\tilde{r}_3(2) = \frac{3c}{4} = r_3^*$. Unfortunately such a procedure would require a great deal of network coordination and more overhead information. Since in general each fixed user would have to inform all links which it utilizes that it is indeed fixed. Thus we will maintain our original algorithm formulation and accept the fact that in general, except for some users at the lowest level, all steady-state user rates are achieved in the limit sense.

4.4 Proof of Convergence

In order to prove convergence we must show:

$$\lim_{N\to\infty} r_i(n) = r_i^* \qquad 1 \leq i \leq M \qquad (4.23)$$

where; r_i^* is the rate assignment for user u_i given by the Centralized Fair Flow control algorithm

The desired proof can be carried out by introducing the notion of rate assignment levels. At the i^{th} iteration of the centralized algorithm a subset of links are found to be the current network bottlenecks {i.e. J^*} . Then each user who utilizes a link $k \in J^*$, and has not been assigned a rate, is assigned a rate which we will call level i, denoted v_i. Property (4) of the centralized algorithm guarantees that $v_i > v_{i-1}$, $i > 0$. We will examine the convergence of user rates in order to increasing level values. Now the formal proof can be presented as follows:

Definition 4.6 Let $W_j(v_i)$ represent the number of users utilizing link j who have a steady-state rate at least as large as v_i in the centralized algorithm.

$$\text{i.e.} \qquad W_j(v_i) = \sum_{\substack{u_k \in \mathcal{U}_j \\ r_k^* \geq v_i}} (1) \qquad (4.24)$$

Definition 4.7 Let $p_j^{v_i}$ represent the ideal desired control value for link j at level i

$$p_j^{v_i} = \frac{1}{W_j(v_i)} \left[c_j - \sum_{\substack{u_k \in \mathcal{U}_j \\ r_k^* < v_i}} r_k^* \right] \qquad (4.25)$$

Note that if the final value of p_j in the centralized algorithm is v_r, then $p_j^{v_r} = \frac{1}{W_j(v_r)} \left[c_j - \sum_{\substack{u_k \in \mathcal{U}_j \\ r_k^* < v_r}} r_k^* \right] = p_j^*$

and $W_j(v_{r+1}) = 0$.

Theorem 4.1

$\forall\, i, \quad \forall\, \epsilon_i > 0, \qquad \exists N \quad \text{s.t.} \qquad \forall\, n > N$

$$\text{(A)} \quad p_j(n) \geq p_j^{v_i} - \frac{\epsilon_i}{2W_j} \qquad \forall_j \tag{4.26}$$

$$\text{(B)} \quad |r_k(n) - r_k^*| < \epsilon_i \qquad \forall k \text{ s.t. } r_k^* \leq v_i \tag{4.27}$$

Theorem 4.1(B) states the convergence of the distributed algorithm.

Proof of Theorem 4.1

The proof proceeds by induction on i

Let $i = 1$;

$$\text{(A)} \quad \text{Note: } W_j(v_i) = W_j \qquad 1 \leq j \leq N \tag{4.28}$$

By (4.22) we have for all $n \geq 0$,

$$p_j(n+1) = p_j(n) + \frac{1}{W_j}\Big[c_j - \sum_{u_k \in \mathcal{U}_j} r_k(n) \Big] , \quad \forall_j \tag{4.29}$$

Since

$$r_k(n) \leq p_j(n) \qquad u_k \in \mathcal{U}_j \;, \quad \forall_j \tag{4.30}$$

$$p_j(n+1) \geq p_j(n) + \frac{1}{W_j}[c_j - W_j\, p_j(n)] = \frac{c_j}{W_j}, \quad \forall_j \tag{4.31}$$

Hence;

$$p_j(n+1) \geq \frac{c_j}{W_j} \geq \min_j \frac{c_j}{W_j} = v_1 \qquad \forall_j \tag{4.32}$$

(which is the desired result).

(B) Assume $r_k^* = v_1$

We must show; (1) $r_k(n) \geq r_k^* - \epsilon_1$

(2) $r_k(n) \leq r_k^* + \epsilon_1$

<u>Case (1)</u>

$$r_k(n) = \min_{m \in L_k} p_m(n) \tag{4.33}$$

However by (4.32)

$$r_k(n) \geq v_1 = r_k^* \quad \forall\ n > o.$$

Hence (1) is true after one step.

<u>Case (2)</u> We would like to show that

$$r_k(n) \leq r_k^* + \epsilon_1 \tag{3.34}$$

Assume that there exists a user

$$u_k \text{ s.t. } r_k(n) > r_k^* + \epsilon_1 \tag{4.35}$$

and find a link $j \in L_k$ s.t. $p_j^* = v_1$.

We show

$$p_j(n+1) < p_j(n) - \frac{\epsilon_1}{2W_j(v_1)} \tag{4.36}$$

as follows.

$$p_j(n+1) = p_j(n) + \frac{1}{W_j(v_1)} \left[c_j - \sum_{u_k \epsilon U_j} r_k(n) \right] \tag{4.37}$$

by (1) $r_k(n) \geq v_1$, $\forall k$, $\forall_n > 0$

$$p_j(n+1) \leq p_j(n) + \frac{1}{W_j(v_1)} [c_j - (W_j(v_1))v_1 - \epsilon_1] \tag{4.38}$$

$$= p_j(n) + \left(\frac{c_j}{W_j(v_1)} - v_1 \right) - \frac{\epsilon_1}{W_j(v_1)} \tag{4.39}$$

However $\frac{c_j}{W_j(v_1)} = v_1$, *thus (4.39) becomes*

$$p_j(n+1) \leq p_j(n) - \frac{\epsilon_1}{W_j(v_1)} < p_j(n) - \frac{\epsilon_1}{2W_j(v_1)} \tag{4.40}$$

which is the desired result .

Hence as long as $r_k(n) \geq r_k^* + \epsilon_1$, p_j is going to decrease by an amount greater than $\left[\frac{\epsilon_1}{W_j(v_1)}\right]$, thus there exists a time when the inequality (4.35) becomes reversed, i.e.

$$\exists N \text{ s.t. } r_k(N) \leq r_k^* + \epsilon_1 \tag{4.41A}$$

Now assume that ε_1 < second smallest $(\frac{c_j}{W_j}) - v_1$.

We can now show that $\forall n \geq N$, conversation k is controlled by a link j with $p_j(n) < r_k^* + \varepsilon_1$ and $p_j^* = r_k^*$, so that $r_k(n) \leq r_k^* + \varepsilon_1$ which is the desired result. Assume link j does it at time n. Then

$$p_j(n+1) = p_j(n) + \frac{1}{W_j}\left[c_j - \sum_{u_m \in \mathcal{U}_j} r_m(n)\right]$$

$$= p_j(n) + \frac{1}{W_j}\left[c_j - \sum_{\substack{u_m \in \mathcal{U}_j \\ m \neq k}} r_m(n) - p_j(n)\right]$$

by (1) $r_m(n) \geq v_1 = p_j^* \quad \forall m, \quad \forall n > 0$, thus,

$$p_j(n+1) \leq p_j(n)\left(1 - \frac{1}{W_j}\right) + \frac{r_k^*}{W_j}$$

Now since link j controls user k, we have:

$$p_j(n) \leq r_k^* + \varepsilon_1 \qquad \text{Thus,}$$

$$p_j(n+1) \leq (r_k^* + \varepsilon_1)\left(1 - \frac{1}{W_j}\right) + \frac{r_k^*}{W_j} = r_k^* + \varepsilon_1 - \frac{\varepsilon_1}{W_j}$$

Hence,

$$\exists N \text{ s.t. } r_k(n) \leq r_k^* + \varepsilon_1 \qquad \forall\ n > N \tag{4.41B}$$

as desired. Moreover the link ℓ controlling k at time n+1 has $p_\ell(n+1) \leq r_k^* + \varepsilon_1$ < second smallest $(\frac{c_j}{W_j})$ and thus $p_\ell^* = v_1$ by (4.32).

In fact the algorithm guarantees that at least one link which has a steady-state control value equal to v_1, will converge to that value in a finite amount of time. This can be shown as follows.

Find a link j and a time N_1 such that

$$p_j(N_1) = \min_m p_m(N_1), \qquad p_j^* = v_1 \tag{4.42}$$

Such a link and time must exist.

<u>Claim:</u> $$p_j(N_1 + 1) = p_j^* \tag{4.43}$$

<u>Proof:</u>

$$p_j(N_1+1) = p_j(N_1) + \frac{1}{W_j(v_1)} \left[c_j - \sum_{k \in \cup_j} r_k(N_1)\right] \tag{4.45}$$

$$= p_j(N_1) + \frac{c_j}{W_j(v_1)} - \frac{W_j(v_1)}{W_j(v_1)} [p_j(N_1)]$$

$$= \frac{c_j}{W_j(v_1)} = p_j^* \tag{4.46}$$

Thus all users who utilize link j will converge to their steady-state rate in a finite number of steps.

Inductive Step

Reset time origin to 0.

Thus assume at $n = 0$, (A) and (B) are satisfied at level i-1 with ϵ_{i-1} as specified below.

We now show that (A) and (B) are satisfied at level i.

(A) Two cases to consider:

$$\text{(I)} \quad p_j^* > v_{i-1} \tag{4.47}$$

$$\text{(II)} \quad p_j^* \leq v_{i-1} \tag{4.48}$$

<u>Case (I)</u> Assume $p_j^* > v_{i-1}$

$$p_j(1) = p_j(0) + \frac{1}{W_j}\left[c_j - \sum_{\substack{u_k \in U_j \\ r_k^* < v_{i-1}}} r_k(0) - \sum_{\substack{u_k \in U_j \\ r_k^* \geq v_i}} r_k(0)\right] \tag{4.49}$$

$$\geq p_j(0) + \frac{1}{W_j}\left[c_j - \sum_{\substack{u_k \in U_j \\ r_k^* < v_{i-1}}} r_k^* - (W_j - W_j(v_i))\epsilon_{i-1} - W_j(v_i)p_j(0)\right] \tag{4.50}$$

$$= \left[\frac{W_j - W_j(v_i)}{W_j}\right] p_j(0) - \left[\frac{W_j - W_j(v_i)}{W_j}\right] \epsilon_{i-1} + \frac{W_i(v_i)}{W_j} \; p_j v_i \tag{4.51}$$

Where the inequality (4.50) holds because by Theorem 4.1(B)

$$r_k(0) \leq r_k^* + \epsilon_{i-1} , \qquad r_k^* \leq v_{i-1}$$

and because $r_k(0) \leq p_j(0)$, $u_k \in U_j$.

Now for notational convenience let

$$\alpha_j(v_i) \overset{\Delta}{=} \left[\frac{W_j - W_j(v_i)}{W_j}\right] \tag{4.52}$$

Iterating

$$p_j(n) \geq \alpha_j^n\,(v_i)\;p_j(0) + \sum_{m=0}^{n-1} \alpha_j^m(v_i)\left[\left(\frac{W_j(v_i)}{W_j}\right)p_j^{v_i} - \alpha_j(v_i)\;\epsilon_{i-1}\right] \tag{4.53}$$

$$= \alpha_j^n(v_i)\;p_j(0) + \left[\frac{1-\alpha_j^n(v_i)}{1-\alpha_j(v_i)}\right]\left[\left(\frac{W_j(v_i)}{W_j}\right)p_j^{v_i} - \alpha_j(v_i)\;\epsilon_{i-1}\right] \tag{4.54}$$

Note: $\dfrac{W_j(v_i)}{W_j} = 1 - \alpha_j(v_i)$ So

$$p_j(n) \geq p_j^{v_i} - \left[\frac{\alpha_j(v_i)}{1-\alpha_j(v_i)}\right]\epsilon_{i-1} + \alpha_j^n(v_i)\left[p_j(0) - p_j^{v_i} + \left[\frac{\alpha_j(v_i)}{1-\alpha_j(v_i)}\right]\epsilon_{i-1}\right] \tag{4.55}$$

We desire $p_j(n) \geq p_j^{v_i} - \dfrac{\epsilon_i}{W_j} \qquad \forall\; n > N'$

So take ϵ_{i-1} such that

$$\left[\frac{\alpha_j(v_i)}{1-\alpha_j(v_i)}\right]\epsilon_{i-1} < \frac{\epsilon_i}{4W_j}\;, \quad \text{for all } j \text{ s.t. } W_j(v_i) \geq 1 \tag{4.56}$$

And take N' s.t.

$$[\alpha_j(v_i]^{N'}[p_j(0) - p_j^*] > \frac{-\epsilon_i}{4W_j}\;, \quad \text{for all } j \text{ s.t. } W_j(v_i) \geq 1 \tag{4.57}$$

Hence (4.55) becomes

$$p_j(n) \geq p_j^{v_i} - \frac{\epsilon_i}{2W_j} \qquad \forall\; n > N' \tag{4.58}$$

which is the desired result.

<u>Case (II)</u> Assume $p_j^* \leq v_{i-1}$ then $p_j^{v_i} = p_j^{v_{i-1}} = p_j^*$ and the desired result is already proved!

Thus (A) is proved for i.

(B) Assume $r_k^* = v_i$ and $n > N'$

$$r_k(n) = \min_{j \in L_k} p_j(n) \geq \min_{j \in L_k} \left(p_j^{v_i} - \frac{\epsilon_i}{2W_j} \right) \tag{4.61}$$

where the latter inequality results from (A).

Lemma 4.1 at the end of this section shows that $p_j^{v_M} \geq v_i$ if $p_j^* \geq v_i$, $v_M \geq v_i$.

$$\text{Thus: } \min_{j \in L_k} p_j^{v_i} \geq v_i \tag{4.62}$$

Using (4.61) and (4.62) we have shown that

$$r_k(n) \geq v_i - \frac{\epsilon_i}{2W_j} \geq v_i - \epsilon_i \tag{4.63}$$

as desired.

Now we need to complete the desired proof by finding an $N > N'$ such that

$$r_k(n) \leq r_k^* + \epsilon_i \quad , \qquad \forall\ n > N \tag{4.64}$$

Assume there exists a user u_k, with $r_k^* = v_i$ s.t. $r_k(n) > r_k^* + \epsilon_i$, $n > N'$ and find a link $j \in L_k$ such that $p_j^* = v_i$. Such a link must exist.

We show:

$$p_j(n+1) \leq p_j(n) - \frac{\epsilon_i}{2W_j} \tag{4.65}$$

$$p_j(n+1) = p_j(n) + \frac{1}{W_j}\left[c_j - \sum_{\substack{u_m \in U_j \\ m \neq k}} r_m(n) - r_k(n)\right] \tag{4.66}$$

By Theorem 4.1 (A) and Case (I); $r_m(n) \geq r_m^* - \epsilon_i/2W_j$, $\forall\ u_m$ s.t. $r_m^* \leq v_i$

$$p_j(n+1) \leq p_j(n) + \frac{1}{W_j}\left[0 + r_k^* + W_j \frac{\epsilon_i}{2W_j} - r_k(n)\right] \tag{4.67}$$

$$\leq p_j(n) + \frac{1}{W_j}\left[\frac{\epsilon_i}{2} - \epsilon_i\right] \tag{4.68}$$

Then

$$p_j(n+1) \leq p_j(n) - \frac{\epsilon_j}{2W_j} \qquad (4.69)$$

Hence as long as $r_k(n) \geq r_k^* + \epsilon_i$

p_j is going to decrease by an amount greater than $\left[\frac{\epsilon_i}{2W_j}\right]$ thus there must exist a time N, when the inequality (4.67) becomes reversed, i.e.

$$\exists N \text{ s.t. } r_k^*(N) \leq r_k^* + \epsilon_i \qquad (4.70A)$$

Now assume that $\epsilon_i < [\frac{v_{i+1} - v_i}{2}]$. At time N user k must be controlled by a link j such that $p_j^* = v_i$.

We now show that $\forall n > N$, user k is controlled by a link j with $p_j(n) < r_k^* + \epsilon_i$ and $p_j^* = r_k^*$, so that $r_k(n) \leq r_k^* + \epsilon_i$ which is the desired result.

$$p_j(n+1) = p_j(n) + \frac{1}{W_j}\Big[c_j - \sum_{\substack{u_m \in \mathcal{J}_j \\ m \neq k}} r_m(n) - p_j(n)\Big]$$

By (4.63) $\quad r_m(n) \geq r_m^* - \frac{\epsilon_i}{2W_j}$, $\quad$ thus

$$p_j(n+1) \leq p_j(n) + \frac{1}{W_j}[r_k^* + W_j (\frac{\epsilon_i}{2W_j}) - p_j(n)]$$

$$= p_j(n) (1 - \frac{1}{W_j}) + \frac{r_k^*}{W_j} + \frac{\epsilon_i}{2W_j}$$

We know: $\quad p_j(n) \leq (r_k^* + \epsilon_i)$

Hence,

$$p_j(n+1) \le (r_k^* + \epsilon_i)\left(1 - \frac{1}{W_j}\right) + \frac{r_k^*}{W_j} + \frac{\epsilon_i}{2W_j}$$

$$= r_k^* + \epsilon_i - \frac{\epsilon_i}{2W_j}$$

So, $p_j(n+1) < r_k^* + \epsilon_i$

as desired. Thus at time n+1, user k is controlled by a link j' with $p_{j'}(n+1) < r_k^* + \epsilon_i$.

Thus,

$$\exists N \quad \text{s.t.} \quad r_k^*(n) \le r_k^* + \epsilon_i \,, \qquad \forall n > N \qquad (4.70B)$$

Combining (4.63B) and(4.70B) we get

$$|r_k(n) - r_k^*| \le \epsilon_i \qquad \forall k \quad \text{s.t.} \quad r_k^* \le v_i$$

and thus convergence is proved. Q.E.D.

We now prove the lemma mentioned previously.

Lemma 4.1 If $p_j^* \geq v_i$

Then $p_j^{v_m} \geq v_i$ for $v_m \geq v_i$

Proof of Lemma 4.1 Assume $p_j^* \geq v_i$, $v_m \geq v_i$

let; $$c_j^* = c_j - \sum_{\substack{u_k \in U_j \\ r_k^* < v_m}} r_k^* - \sum_{\substack{u_k \in U_j \\ r_k^* \geq v_m}} r_k^* \geq 0 \tag{4.71}$$

$$c_j - \sum_{\substack{u_k \in U_j \\ r_k^* < v_m}} r_k^* \geq \sum_{\substack{u_k \in U \\ r_k^* \geq v_m}} r_k^* \geq v_m W_j(v_m) \geq v_i W_j(v_m) \tag{4.72}$$

$$\frac{1}{W_j(v_m)} \left[c_j - \sum_{\substack{u_k \in U_j \\ r_k^* < v_m}} r_k^* \right] \geq v_i \tag{4.73}$$

Hence
$$p_j^{v_m} \geq v_i \qquad \forall \quad v_m \geq v_i \tag{4.74}$$

which is the desired result .

4.5 The Jaffe Scheme

While this research was in progress J.M. Jaffe [14] published interesting results on a flow control algorithm. The objectives of Jaffe's scheme are quite similar to ours, as presented in section 4.2, however, he introduces the following new idea.

Instead of choosing user rates in a fair manner such that;

$$(c_j^* - f_j^*) = 0 \qquad \forall_j \tag{4.75}$$

which our algorithm does,

select user rates in a fair manner such that

$$(c_j^* - f_j^*) = (\max_{k \varepsilon L_j} r_k^*)/x \tag{4.76}$$

where x is a positive constant.

There are basically two reasons for using (4.76),

(i) Assume a new user, say u_{new}, initializes a call. We know that the rate allocated to it must be determined by the bottleneck link on its route, and thus we have

$$r_{new}^* \leq \max_{k \varepsilon L_j} r_k^* \quad , \qquad \forall_j \varepsilon L_{new} \tag{4.77}$$

Now if $x = 1$ in (4.76) we have

$$(c_j^* - f_j^*) = \max_{k \varepsilon L_j} r_k^*$$

Hence we can accommodate user u_{new} without causing $c_j^* < f_j^*$ for any $j \varepsilon L_{new}$.

(ii) Using (4.76) protects the network against percentage changes in each user's rate due to transient conditions. Thus if a user increases its rate by a factor (1/x), the inequality $c_j^* \geq f_j^*$ still applies.

Jaffe presents an algorithm to compute the user rate assignment. It is essentially the centralized algorithm presented in section 4.2, except that

$$p_j(i+1) = \left[\frac{c_j - \hat{f}_j(i)}{1/x + \hat{w}_j(i)}\right]$$

where $\hat{f}_j(i)$ = sum of the rates of the users on link j that have been fixed before the ith iteration

$\hat{w}_j(i)$ = the number of users which have not been fixed by the ith iteration.

Of course this algorithm has the same finite convergence property as all centralized algorithms. In order for this algorithm to be distributed, the following is required:

(1) Before executing the algorithm the rate of each user must be set to 0.

(2) Each link j must keep track of $\sum$(rates fixed), and the number of users unfixed.

(3) After every step until its rate is fixed, each user must inform all links on its route of its rate.

Restriction (1) is a major drawback of Jaffe's algorithm because it essentially implies that each time a new user enters the system, the rate of each user already active in the network must be set to zero in order to carry out the algorithm. Clearly this is not desirable for a packet voice network.

Restrictions (2) and (3) require a lot of cooperation between each user and the links in its route, but still allow the algorithm to be decentralized in the sense that the rate of each user is determined only by the links in its route.

To modify the algorithm developed in this chapter to use Jaffe's objective (4.76) instead of (4.75), we simply need to introduce the notion of a fictitious user u_{F_j} for each link j, who utilizes only link j, with rate : $r_{F_j}(N) = [p_j(N)/x]$. By doing this we are essentially reserving enough capacity on each link such that after convergence each link j is able to accommodate one new user at rate p_j/x without having $f_j^* > c_j^*$. Hence we can alter our algorithm as follows

$$p_j(\text{new}) = p_j(\text{old}) + \left[\frac{1}{w_j + \frac{1}{x}}\right][c_j - f_j(\text{old}) - p_j(\text{old})/x]$$

$$\text{for } 1 \leq j \leq N \qquad (4.78)$$

Note that our algorithm under these changes still remains completely decentralized. In addition the algorithm as before is incremental (i.e. the algorithm converges from any initial $\vec{p}(0)$) and thus doesn't suffer restriction (1) of Jaffe's algorithm.

As we will see in the next chapter the use of criteria (4.76) instead of (4.75) leads to improved performance of the Fair Flow control algorithm and thus Jaffe's idea is of significant value.

CHAPTER V

SIMULATION AND RESULTS

In this chapter a computer program is developed to simulate a general packet-switched network. The program monitors every packet generated from each source, thus allowing us to obtain detailed measurements.

We first develop the program. Then after the specific network model is chosen, we examine the performance of the algorithms developed in chapter IV.

5.1 Simulation Model

We would like to model the behavior of a packet-switched voice network so that a simulation program can be developed to determine the performance of the decentralized Fair Flow control algorithms presented in Chapter IV. To develop our model we must characterize both the end-to-end control mechanism and the source operation.

5.1.1 End-to-End Control Mechanism

Each link j in the network will measure its flow over a time period denoted by TOBS(j) and then compute its new control value using the appropriate update equation. The necessary control information can be transmitted through the network by the following scheme.

Each source generates either voice or control packets. The structure of these packets is illustrated in Fig. 5.1

Header	Forward Control Information (FO)	Feedback Control Info (FE)	Voice Information

Fig. 5.1(a) Voice Packet Format

Header	(FO)	(FE)

Fig. 5.1(b) Control Packet Format

Let us denote the forward control information by FO and the feedback control information by FE. Furthermore, let us assume that each source u_i maintains two control variables;

(1) SCNTRL (i) - which represents the current allowable total transmission rate for source i

(2) DCNTRL(i) - which represents the feedback rate for source u_i's partner across the network.

Then the control mechanism works as follows. The value of FO is initially set to infinity. Each link along the route to the destination compares FO with its current link control value and if the latter is less than FO, it substitutes that value for FO, otherwise FO is left unchanged. When the packet reaches its destination, the FO field will indeed contain the minimum current link control value over all links in the packet's route. The destination will then

(1) read FE to learn the rate at which the network can support its transmission

(2) set its SCNTRL = FE

(3) read FO

(4) set its DCNTRL = FO.

By carrying out the procedure discussed above the necessary control information will be continually exchanged between the two ends of the conversation.

5.1.2 Source Operation

Each conversation in the network consists of a pair of users which alternate between talkspurt and silence modes. We assume that if users u_1 and u_2 comprise a conversation, the completion of a talkspurt period of user u_1 is always followed by a talkspurt period of user u_2, and vice versa. Thus we do not allow a talkspurt period of a particular user, say u_1, to be followed by a silence period for both u_1 and u_2 and then another talkspurt period for u_1. As a result our model is not an exact representation of conversational speech, however, it is still a reasonable and acceptable model.

The behavior of each source at any given time is dependent upon which mode it is currently in.

A. Silence Mode

If a user is in silence mode we assume that it will generate fixed length control packets periodically with a time period denoted by ICPINC. The length of the control packet will be denoted by LENCON

B) Talkspurt Mode

If a user is in talkspurt mode we assume it will generate voice packets at an average rate of 50 (packets/sec.), or equivalently a voice packet is generated approximately every 20 milliseconds (which is the value most commonly used in practice). In order to introduce some additional randomness the actual time period between voice packet generation will be represented by a sample from a uniform distribution defined over the range [18,22] milliseconds.

The length of the generated voice packet is dictated by the source's current control value, i.e. SCNTRL. Thus we determine the length of the entire voice packet by the relationship

$$\text{Total voice packet length} = \left[\frac{\text{SCNTRL (bits/sec.)}}{\text{50 (packets/sec.)}}\right] = \left[\frac{\text{SCNTRL}}{50}\right] \text{ (bits/packet).} \tag{5.1}$$

Since each voice packet must contain a fixed number of bits (for header and control information), the actual rate at which voice information is being coded is given by the relationship:

$$\text{Voice Coding Rate} = \text{SCNTRL} - 50 \text{ (packets/sec) LENCON(bits/packet)} \tag{5.2}$$

$$= [\text{SCNTRL} - 50 \text{ (LENCON)}] \text{ (bits/sec)} \tag{5.3}$$

To model the creation of a talkspurt period we must first determine the number of voice packets a user will have to transmit. This can be determined by finding the particular duration of the talkspurt period. The probability distribution of conversational talkspurt and silence durations were measured by Brady [15], and these measurements can be used as the basis for a statistical model for talkspurt duration. The standard statistical model used is to represent talkspurt duration as an exponential random variable with mean value equal to 1.2 seconds. Thus if τ represents a sample from this distribution then the number of packets a user will have for its talkspurt is

Number of voice packets for talkspurt =

$$\text{Integer } [50 \text{ (packets/sec) } \tau\text{(seconds/talkspurts)}] + 1 \tag{5.4}$$

where, τ is a sample from an exponential distribution with mean value equal to 1.2 seconds.

Having completed our simulation model development we are now ready to proceed to the development of the simulation program.

5.2 Simulation Program

A simulation program (see Appendix) has been developed on the basis of Section (5.1). The program was constructed by esentially noting that there are five fundamental events that can take place in the network. Then by determining, scheduling, and executing these events in chronological order we were able to produce the desirable model. The five events are discussed below.

(1) Absorption - A packet is absorbed when it finally reaches its destination. The destination receiver then reads the control information in the packet and updates its control parameters. Furthermore, if the packet was the last voice packet of a talkspurt then the destination generates its talkspurt.

The packet is then removed from the presence of the network after a small receiver processing delay, denoted TPROC (1).

(2) Arrival - A packet arrives at a link which it is to be transmitted on. The size of the link queue is increased by 1. There are two cases to consider. When the packet arrives at link j it either

(a) finds the queue for link j empty

(b) finds the queue for link j non-empty.

If (a) occurs, then after a small link processing delay, denoted TPROC(2), the packet is scheduled for transmission.

If (b) occurs then packet is placed at the end of the link queue.

(3) Transmit - When a packet is to be transmitted on link j the following sequence of steps must be executed.

(a) Determine the next location for the packet. The next location will either be another link or the destination receiver.

(b) Determine the amount of transmission time it will take to transmit the packet.

(c) Perform appropriate data manipulation upon the packet's control data field, using link j's current control value.

(d) Using (b), the value for link j's propagation delay, denoted TPROP(j), and the link processing delay, denoted TPROC(3), schedule the packet at the next location which was determined in (a).

(e) Eliminate the presence of the packet from the link j queue after a time period equal to [packet transmission time + TPROP(j)] has elapsed.

(4) Packet Generated at Source

A packet generated at each source u_i is either:

(a) a control packet if the source is currently in silent mode

(b) a voice packet if the source is currently in talkspurt mode.

For (a) the following sequence of steps must be executed;

(i) The packet length is set to a constant value equal to LENCON.

(ii) The forward control information field is set to ∞ .

(iii) Feedback control information (i.e. DCNTRL(i)) is placed in the packet for use by the destination.

(iv) The packet is scheduled to arrive at the first link in its route after a source processing delay denoted TPROC(4).

For (b) the following sequence of steps must be executed:

(i) Determine whether or not the packet is the last voice packet of the source's talkspurt.

(ii) Determine the length of the packet as a function as per (5.1)

(iii) Schedule the packet to arrive at the first link in its route after a source processing delay of TPROC(4).

(iv) Decrement the number of packets remaining in talkspurt.

(5) Link Control Value Update - Each link j measures its average flow over the observation period TOBS(j). At the end of this period it updates its control value according to the appropriate link control value update equation.

The next link control value update is then scheduled for TOBS(j) seconds in the future.

The simulation program uses two tables, ETABLE and PACKET, to continually execute the five different types of events in proper chronological order. In addition provision is made to periodically compute statistical information regarding the links and sources in the network.

5.3 Network Model for Simulation Program

In order to have a basis of comparison, we choose to select a network model which resembles that used in the (LL) simulation discussed earlier. However the (LL) network model [10] consisted of 800 sources, which would create a tremendous computational and booking load for our simulation program since we monitor all packets generated by all sources. Thus we chose to scale the network by a factor of (1/10). Furthermore the (LL) network model considered traffic flow in only one direction whereas to consider to examine the effects of important delay parameters (which the (LL) simulations did not) we must consider two-way traffic flow, as discussed in section 5.1. One way of modifying the (LL) model to allow two-way traffic flow without adding any additional links is to view all sources as being at the same location. The final network that will be used in our simulation is illustrated in Figure 5.2.

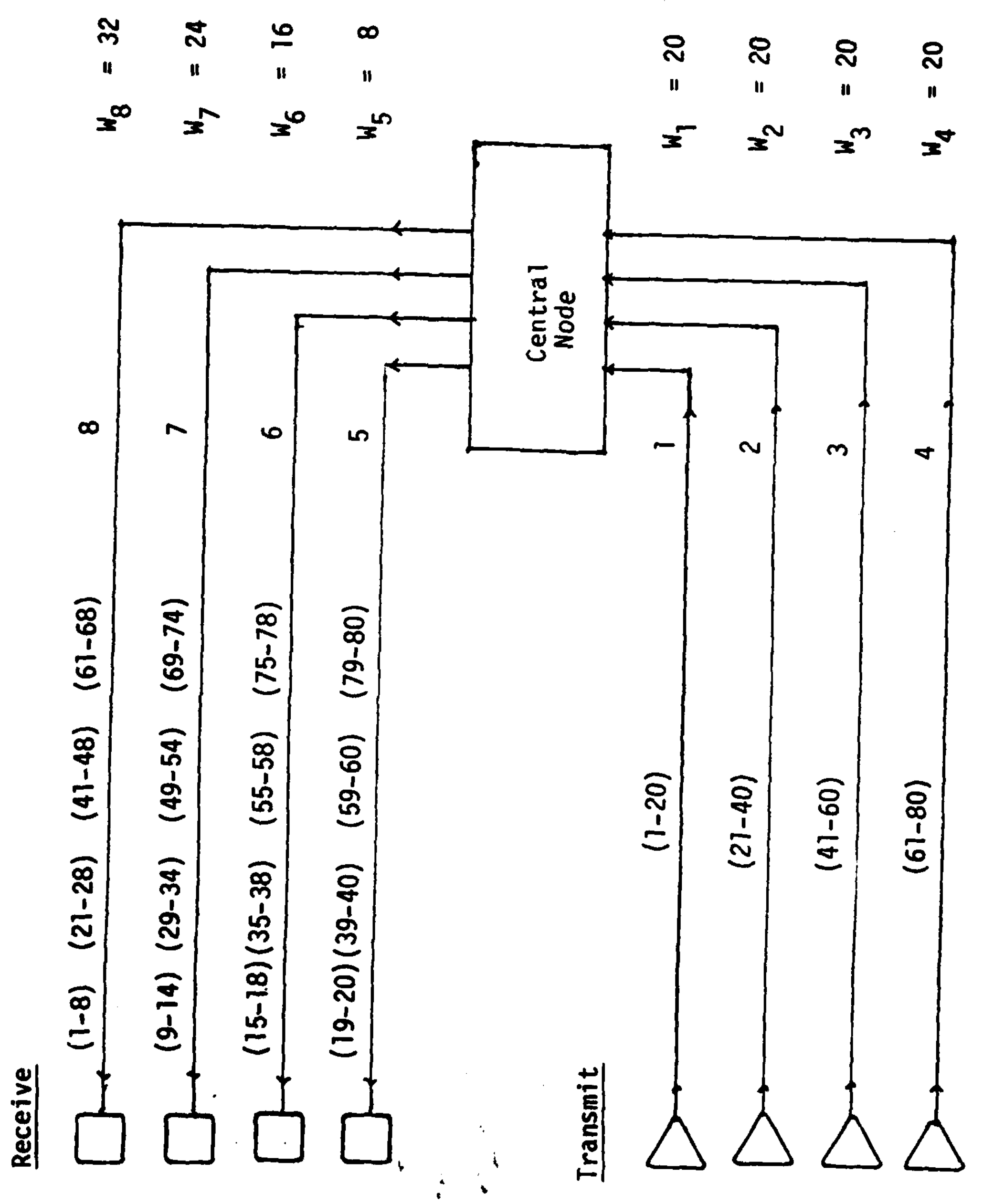


Figure 5.2 Simulation Network Model

Each used i is paired with user (81-i), for $1 \leq i \leq 40$. Ideally we would like no correlation between the set of links which user i utilizes and the set of links which its partner, namely user (81-i), utilizes since this would be the case if we had indeed incorporated additional links to support ideal two-way traffic. However, notice from Fig. 5.2 that the user pairs:

user 32 $\Longleftrightarrow$ user 49

user 31 $\Longleftrightarrow$ user 50

user 30 $\Longleftrightarrow$ user 51

user 29 $\Longleftrightarrow$ user 52

are indeed correlated since each pair uses link 7. Thus with our network model we do not achieve complete link independence of user pairs, but because only link 7 is involved, and only 8 users out of a total of 24 on that link are involved,the consequence should be negligible.

It also should be noted that in the original (LL) network the link capacities were all equal to 0.40 mbits/sec, however since we must scale down by a factor of 10, the true capacities of all link in Fig. 5.2 will be set equal to 40 Kbits/sec.

5.4 Results

The parameters used in all simulation runs were chosen to be consistent with respect to current technology. They are as follows:

Overhead Number of bits per packet (LENCON) = 10 bits

Control packet intergeneration time (ICPINC) =100 milliseconds

Receiver processing time for packet Absorbtion (TPROC(1))= 5×10^{-4} sec.

Link processing time for packet Arrival (TPROC(2)) = 5×10^{-4} sec.

Link processing time for packet Transmission (TPROC(3)) = 1×10^{-4} sec.

Source processing time for packet Generation (TPROC(4)) = 5×10^{-4} sec.

We used as the effective link capacity of link i , (0.8) times the true capacity of link i. Thus the effective capacity of each link i was equal to 32 k bits/sec., i.e.

$$c_i = 32 \text{ kilobits/sec} \qquad \text{for } 1 \leq i \leq 8$$

Furthermore the propagation delay of each link i was taken to be 3 milliseconds, i.e.

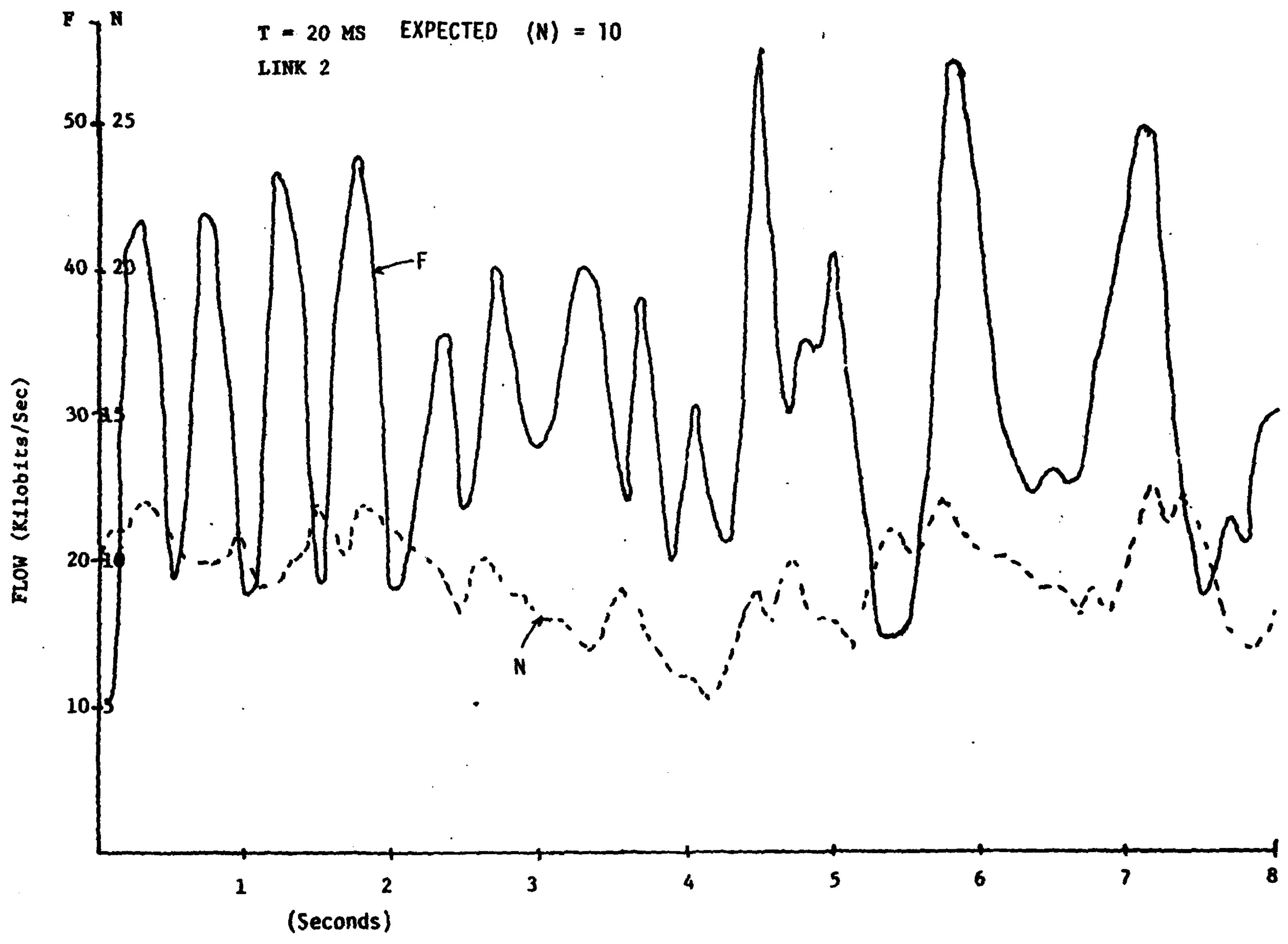
$$\text{TROP}(i) = 3 \text{ milliseconds for } 1 \leq i \leq 8.$$

For simplicity, we assumed that the time period between control updates for all links will be identical, however, the link control updates are not carried out synchronously in the program. Thus let us denote $T = \text{TOBS}(i)$, for $1 \leq i \leq 8$. The parameter T is the primary variable of concern since altering T over a range of values can dramatically change the performance of the algorithm.

<u>T = 20 ms</u>

Shown in Figure 5.3 is a plot of the average flow (averaged over the previous 100 ms) on link 2 as a function of time. The dotted line on the graph portrays the number of active speakers on link 2, denoted by N, at a given time. The flow always lags N because of the presence of delay.

It is clear from Fig. 5.3 that choosing T = 20 ms leads to oscillation since a small change in the number of active speakers can lead to a large change in the resultant link flow. The occurence of oscillation is due to the fact that although link control updates are performed every 20 ms, the time period between source rate updates is equal to at least 100 ms (i.e. ICPINC). Thus if link j controls user u_i, then $r_i(n) \approx p_j(n-4)$. As a result, source rates are being assigned on the basis of old data which does not reflect the current network status. In this case our model is no longer valid and the control update algorithm breaks down. In conclusion T should be chosen more comparable to the maximum of the round trip delay and the control packet intergeneration time.



F
N
T = 20 MS EXPECTED (N) = 10
LINK 2
50
25
40
20
30
15
20
10
10
5
FLOW (Kilobits/Sec)
F
N
1
2
3
4
5
6
7
8
(Seconds)

Figure 5.3

T = 100 ms

A. Quasistatic Behavior

By examining Fig. 5.4 we see that the algorithm basically does achieve its goal of keeping the average flow around the effective link capacity (i.e. 32 kilobits/sec). We also observe over periods of time where the number of users remains more or less constant, that the algorithm takes about five steps (i.e. 500 ms) to yield the desired link flow. This can be explained by noting that link 2 usually has control only over its users which also either link 5 or 6 and as a result we expect link 2 to be controlling about 3 active users only, leading to decay rate $\alpha_2(v_3)$ given by (4.52) of about (14/20), and $[\alpha_2(v_3)]^5$ is negligible.

B. Dynamic Behavior

To obtain a basis of evaluation of the dynamic (short term) behavior of the algorithm we also performed a simulation run in which users were assigned fixed rates as determined by the Centralized Fair Flow control algorithm.

Comparing Fig. 5.4 and 5.5 we can observe the important advantage gained by incorporating dynamic flow control. The advantage is that in the NO CONTROL case the flow always follows number of activer users exactly, whereas in the control case the algorithm smooths out fluctuations in the number active users. For example, examine Fig. 5.4 for the time interval [2, 2.8] seconds.Note that the number of activer users went from 8 to 16 (100% increase) whereas the flow only went from 27 to 40 kbits/sec. (48% increase). In contrast examine Fig. 5.5 for the time interval [2.2,2.6] seconds. Note that the number of active users went from 10 to 13 (33% increase), the flow went from 32 to 45 (40% increase), the increase is not 1 - 1 since all users do not have the same rate.

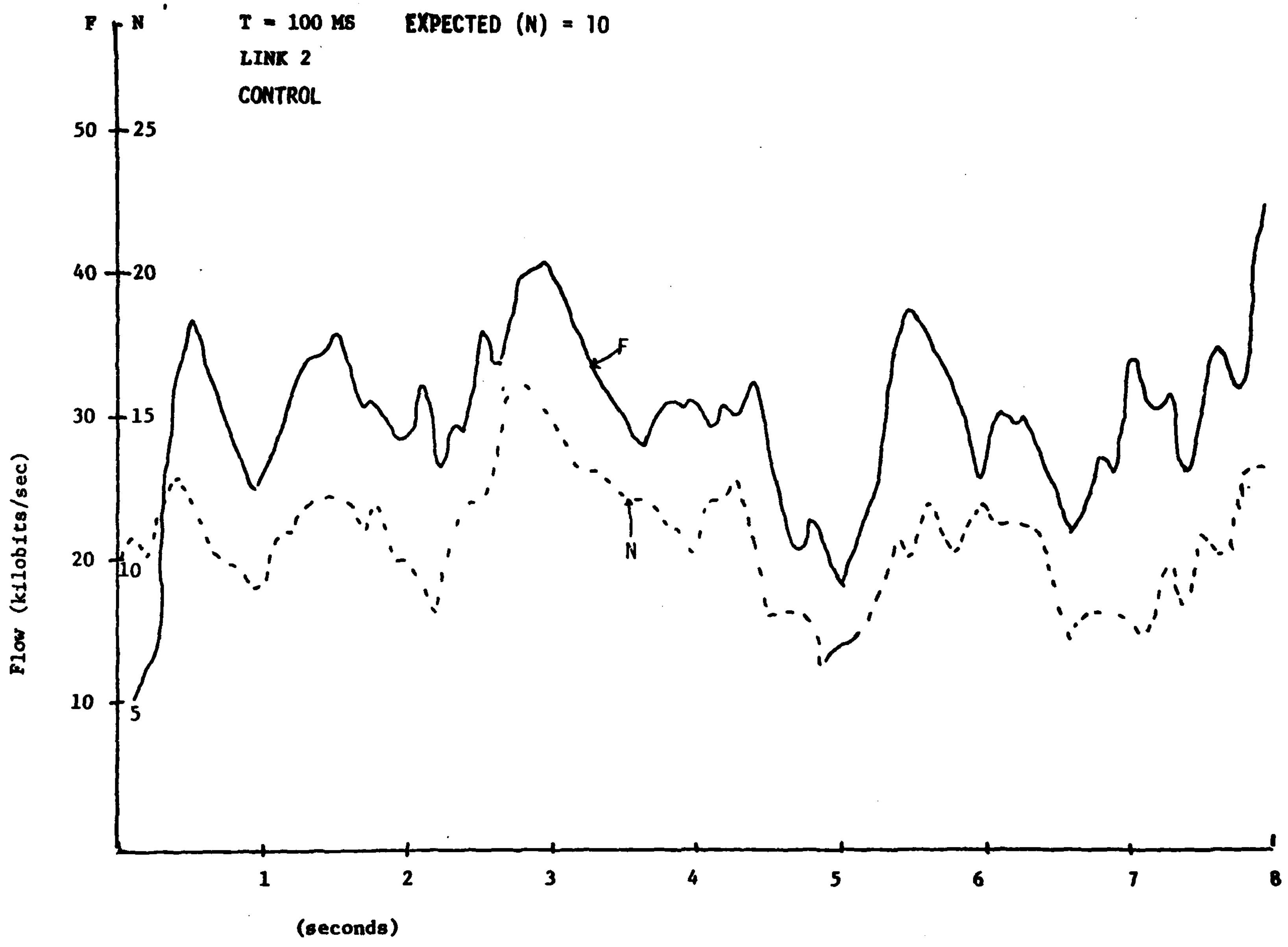

T = 100 MS
EXPECTED (N) = 10
LINK 2
CONTROL
F
N
50
25
40
20
30
15
20
10
10
5
Flow (kilobits/sec)
1
2
3
4
5
6
7
8
(seconds)

Figure 5.4

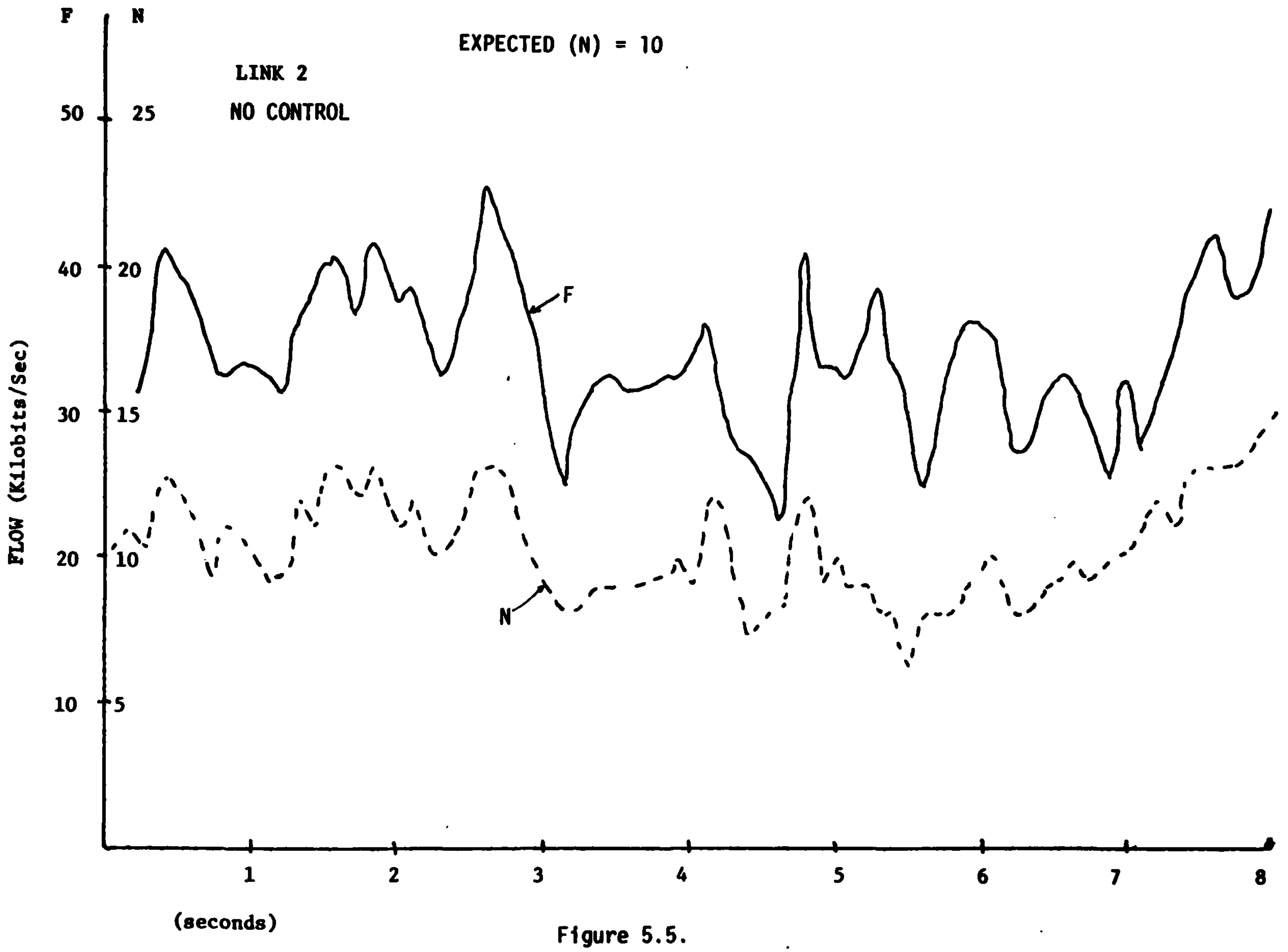
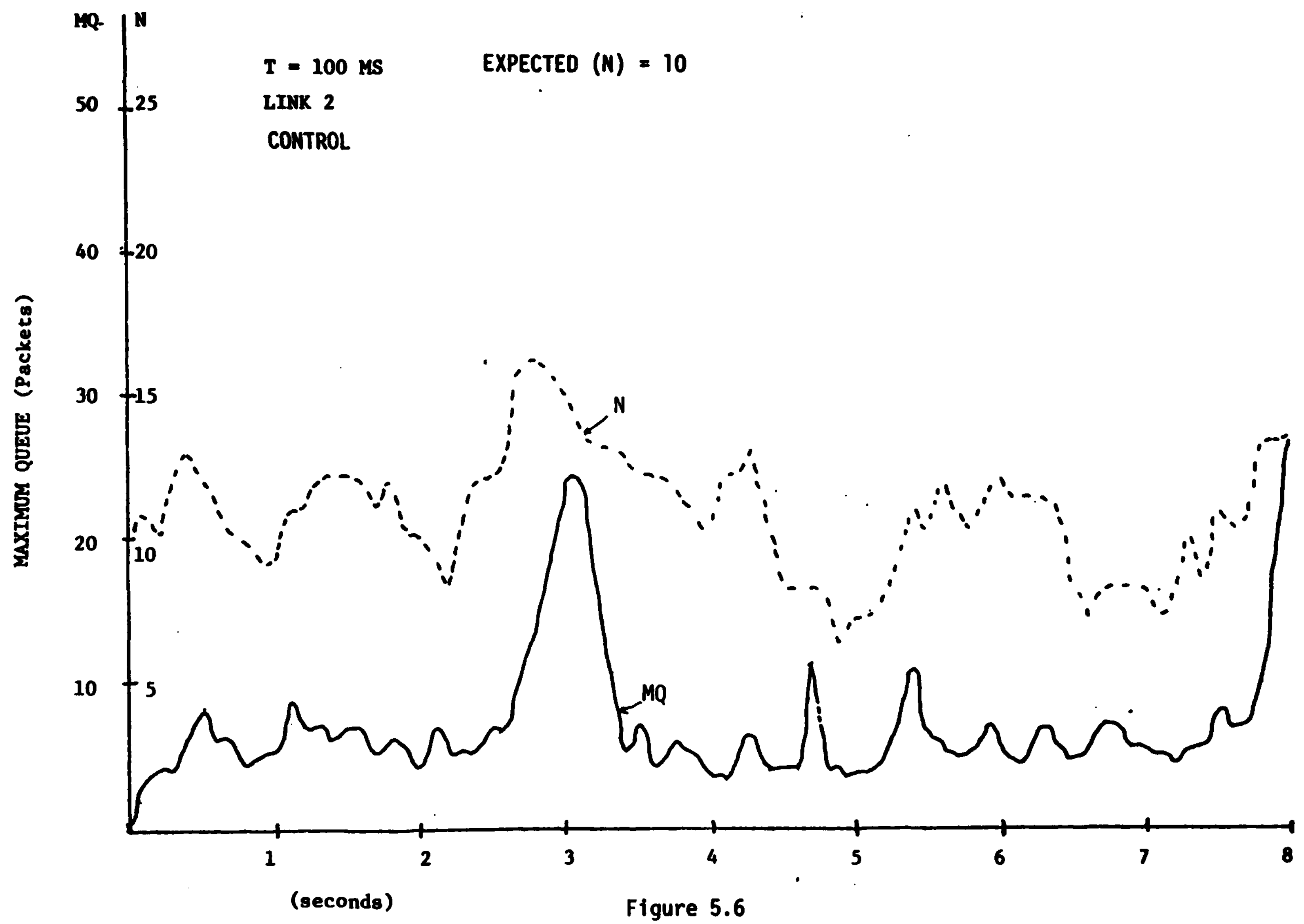

F
N
EXPECTED (N) = 10
LINK 2
NO CONTROL
50
25
40
20
30
15
20
10
10
5
FLOW (Kilobits/Sec)
F
N
1
2
3
4
5
6
7
8
(seconds)

Figure 5.5.

Another major attribute of using dynamic rate assignment is clearly illustrated by comparing Fig. 5.6 and Fig. 5.7, where the quantity maximum queue size refers to the maximum over the preceding 100 ms. Note that in the NO-CONTROL case a slight increase in the number of active users could result in large queues because the coding rate of all users who utilize link 2 is based on the assumption that link 2 always will have ten active speakers at any given time, resulting in an average rate of 3.2 kilobits/sec/user. Thus when N rises above 13, the coding rate will be too high for this condition and the desired link flow will exceed the true link capacity (40 kilobits/sec), hence resulting in large queues. However in the CONTROL case when N increases the coding rate of those users on link 2 which are being controlled by link 2 will decrease to compensate. Thus only in the cases where the number of active speakers increases dramatically (e.g. 50%) over a short period of time will queues be able to build up. Furthermore, note that even in those cases the maximum queue size decreases rapidly compared to duration of time which N remains large. To substantiate the preceding discussion examine Fig. 5.6 for the time interval [2.5,3.3] seconds.

The performance of link 8 is shown in Fig. 5.8 - 5.11. The conclusions drawn from examining these figures are consistent with those for link 2.



MQ
N
T = 100 MS
LINK 2
CONTROL
EXPECTED (N) = 10
50
40
30
20
10
25
20
15
10
5
MAXIMUM QUEUE (Packets)
N
MQ
1
2
3
4
5
6
7
8
(seconds)

Figure 5.6

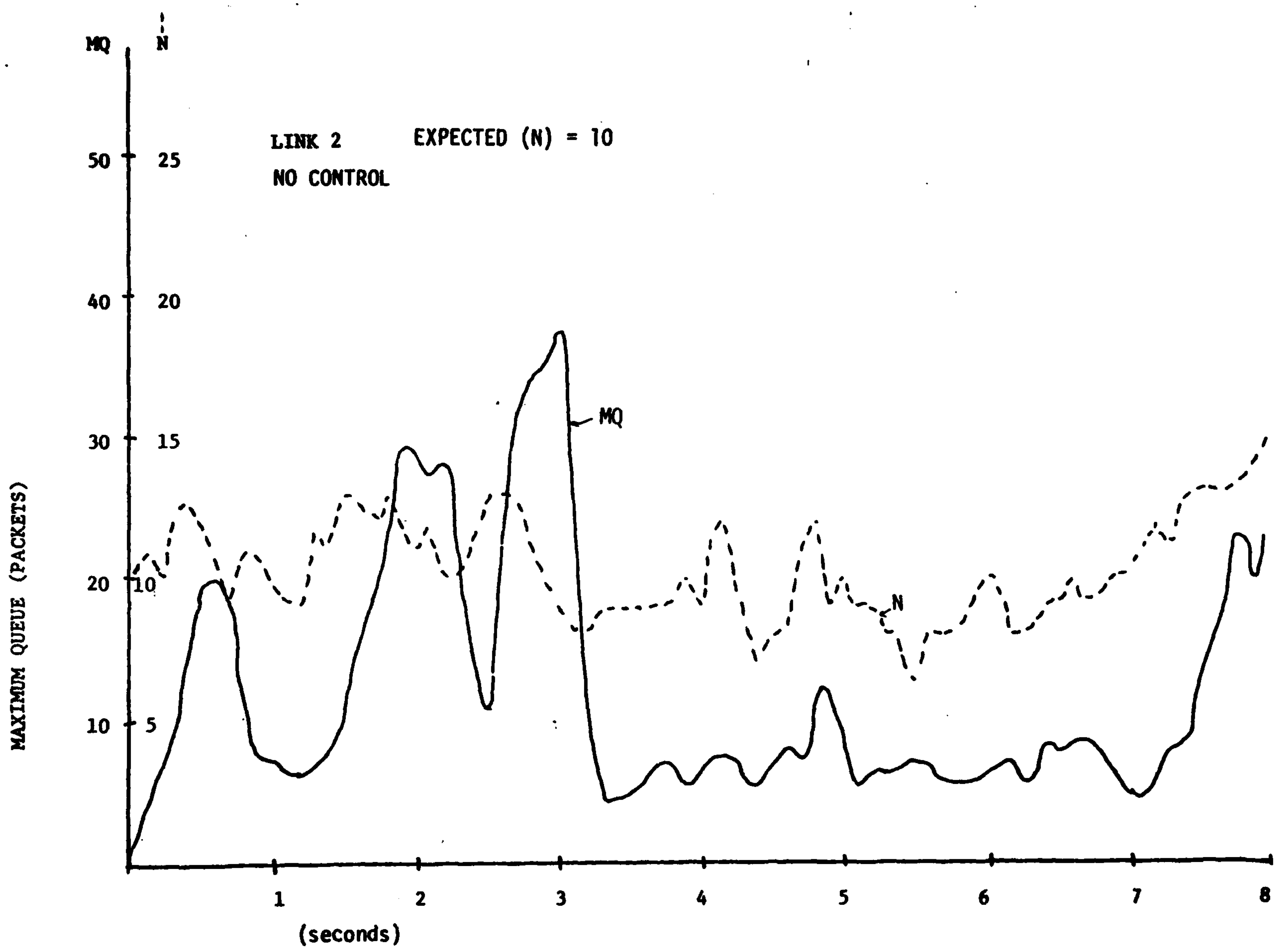

MQ
N
LINK 2
NO CONTROL
EXPECTED (N) = 10
50
40
30
20
10
25
20
15
10
5
MQ
N
MAXIMUM QUEUE (PACKETS)
1
2
3
4
5
6
7
8
(seconds)

Figure 5.7

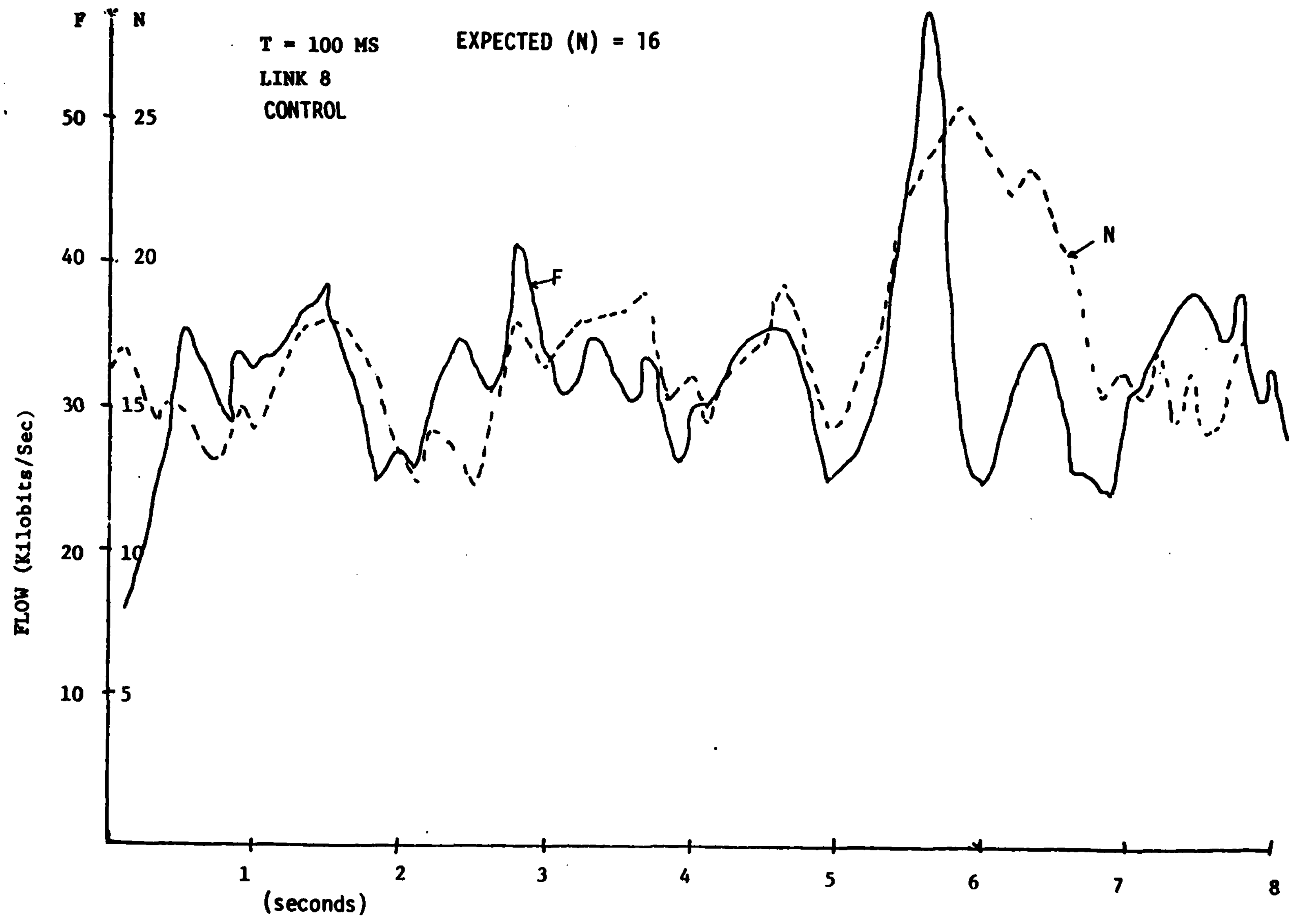

F
N
T = 100 MS
LINK 8
CONTROL
EXPECTED (N) = 16
50
40
30
20
10
25
20
15
10
5
FLOW (Kilobits/Sec)
F
N
1
2
3
4
5
6
7
8
(seconds)

Figure 5.8

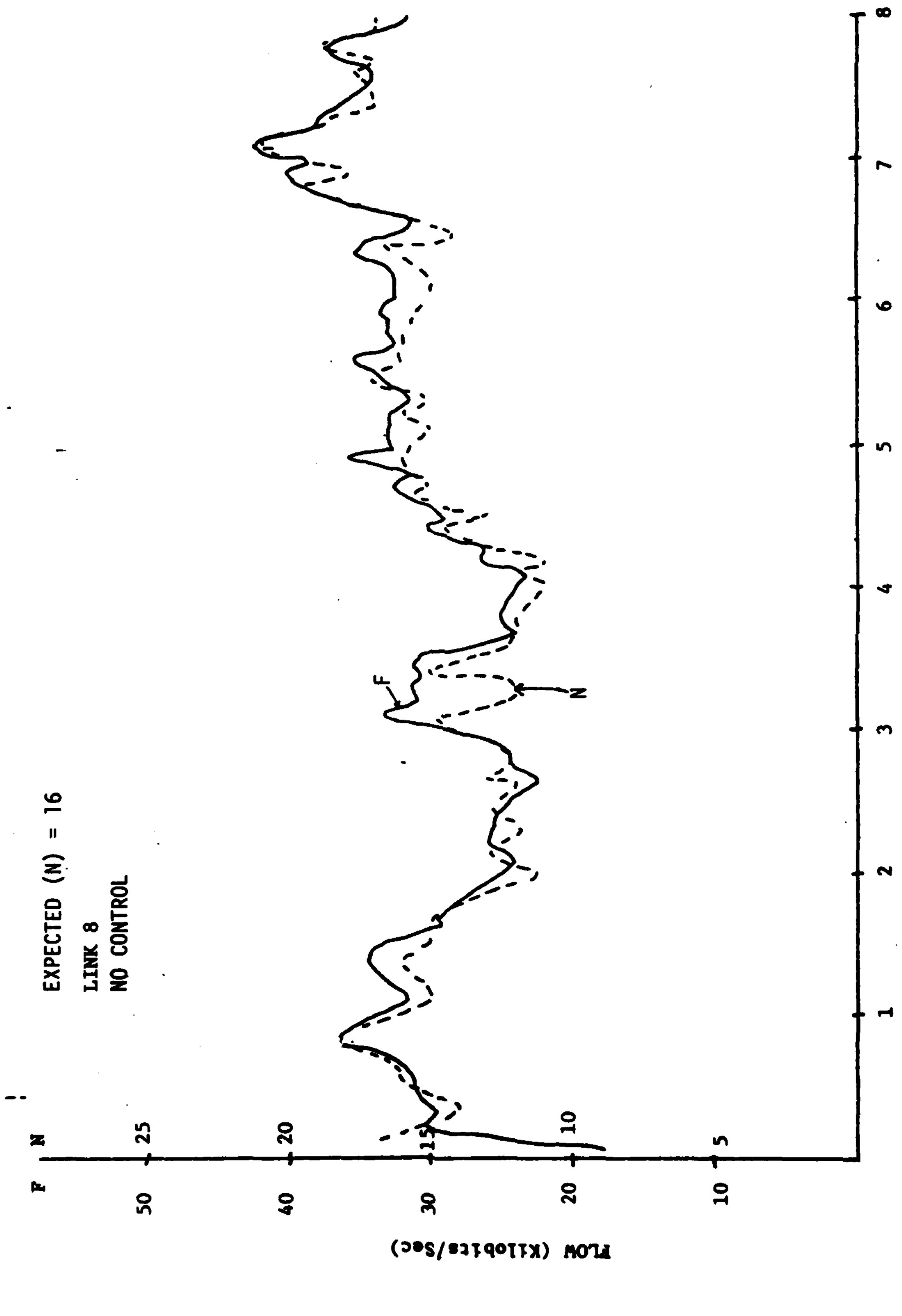
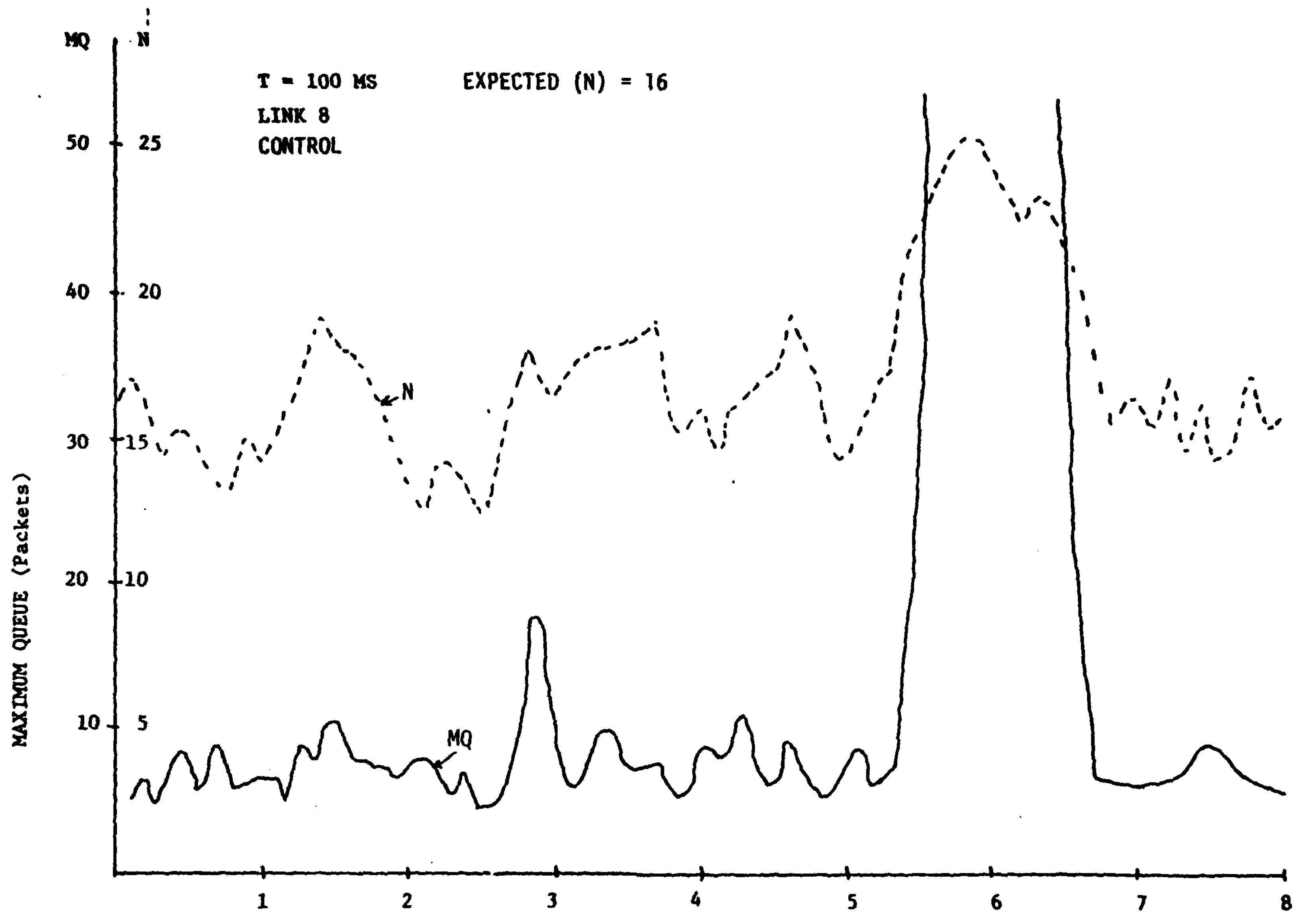
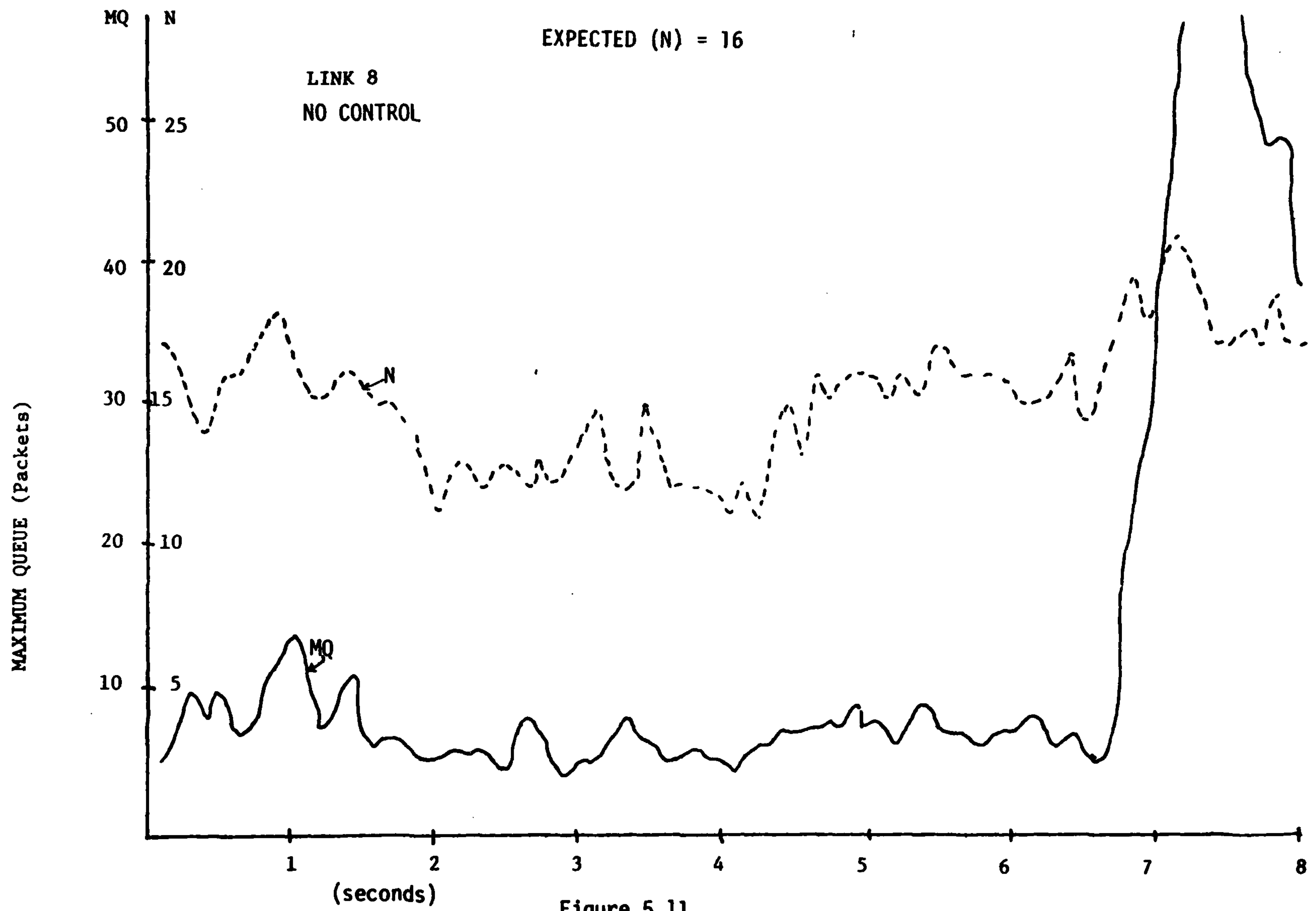

EXPECTED (N) = 16
LINK 8
NO CONTROL
F
N
50
40
30
20
10
25
20
15
10
5
1
2
3
4
5
6
7
8
FLOW (Kilobits/Sec)
(Seconds)

Figure 5.9



MQ
N
MAXIMUM QUEUE (Packets)
T = 100 MS
LINK 8
CONTROL
EXPECTED (N) = 16
50
40
30
20
10
25
20
15
10
5
N
MQ
1
2
3
4
5
6
7
8
(seconds)

Figure 5.10



MQ
N
EXPECTED (N) = 16
LINK 8
NO CONTROL
50
25
40
20
30
15
20
10
10
5
N
MQ
MAXIMUM QUEUE (Packets)
1
2
3
4
5
6
7
8
(seconds)

Figure 5.11

C. Jaffe's Criteria

We extended our algorithm to incorporate Jaffe's criteria as discussed in section 4.5 and ran the simulations over using T=100 ms. The results for link 2 are shown in Figs. 5.12 - 5.13 and those for link 8 in Figs. 5.14 and 5.15. Comparing these results with those of our original algorithm we find that by incorporating Jaffe's idea we apparently eliminate the occurance of large queues. A possible explanation for this behavior can be given by first computing some simple averages.

Control (without Jaffe's criteria)

Link 2:

Desired flow = 32 (k bits/sec)

Average total coding rate per user = $(\frac{32}{10})$ = 3.2 (kb/s)

Average control value = 8 (kb/s)

Excess capacity = (40-32) = 8 (kb/s)

Link 8

Desired flow = 32 (kb/s)

Average total coding rate per user = $(\frac{32}{16})$ = 2 (kb/s)

Average control value = 2 (kb/s)

Excess capacity = 40-32 = 8 (kb/s)

Control (with Jaffe's criteria)

Link 2:

Desired flow = $(\frac{10}{11}) \approx 29$ (kb/s) (*)

Average total coding rate per user $\approx (\frac{29}{10}) = 2.9$ (kb/s)

Average control value $\approx$ 4.2 (kb/s)

Excess Capacity = 40 - 29 = 11 (kb/s)

(*) where the factor $(\frac{10}{11})$ comes from the addition of the fictitious user

Link 8:

Desired Flow = $(\frac{16}{17})$ (32) $\approx$ 30 (kb/s)

Average total coding rate per user $\approx \frac{30}{16} \approx 1.9$ (kb/s)

Average control value $\approx$ 1.9 (kb/s)

Excess capacity $\approx$ 40-30 $\approx$ 10 (kb/s)

Link 2 has users which are controlled by link 2 as well as users which are controlled by link 7 or 8, whereas link 8 has control over all its users. Now using our proceeding results we can compute the average number of additional (i.e. above E(N)) users which can be accommodated on link 2 without causing $f_2 > 40$ (kb/s) and the average number of additional users which can be accommodated on link 8 without causing $f_8 > 40$ (kb/s).

<u>Link 2 (without Jaffe)</u>

<u>Best case</u> - all new users are controlled by link 7 or 8

Average number of additional users $= \frac{8}{2} = 4$

<u>Worst Case</u> - all new users are controlled by link 2.

Average number of additional users $= \frac{8}{8} = 1$

<u>Link 2 (with Jaffe)</u>

<u>Best Case</u> - all new users are controlled by link 7 or 8

Average number of additional users $= \frac{11}{1.9} \approx 5$

<u>Worst Case</u>- all new users are controlled by link 2

Average number of additional users $\approx \frac{11}{4.2} \approx 2$

Thus by using Jaffe's criteria we are able to improve the worst case by a factor of 2. This fact can significantly improve performance (in terms of maximum queue size) since in the control case without Jaffe's criteria we can only accommodate one additional user who uses link 2 and 5 (there are only 2 total). Thus if the two users happen to become active at approximately the same time (i.e. within about 1/2 second) the desired link flow will be about 48 kb/s resulting in the rapid development of a large queue. This event will occur on the average about once every 4 seconds (e.g. see Fig. 5.6),hence it is not negligible.

Link 8 (without Jaffe)

All users are controlled by link 8

Average number of additional users $= \frac{8}{2} = 4$

Thus if N(link 8 without J) $< 20 \Longrightarrow f_8 < 40$ (kb/s) I

Link 8 (with Jaffe)

All users are controlled by link 8

Average number of additional users $\approx \frac{10}{1.9} \approx 5.$ II

Thus if N(link 8 with Jaffe) $< 21 \Longrightarrow f_8 < 40$ (kb/s)

The preceding results do not offer a clear explanation for link 8. Figure 5.15 illustrates $N \leq 20$ hence (I) is satisfied and as expected only small queues develop. In contrast if we examine the result for the control case (without Jaffe) i.e. Fig. 5.10 we find in fact that N actually went up to 25 and as a result the link flow greatly exceeded the true capacity (see Fig. 5.8) and thus large queues developed. Thus we conclude that good performance resulted in the Jaffe case because we were lucky, in the sense that N never rose above 20.

In summary, we feel Jaffe's criteria will improve performance to a certain extent. However, because we performed only one simulation run using this criteria, we are unable to draw any final conclusions. Thus more and/or longer simulation runs should be performed in order to gain more quantitative results.

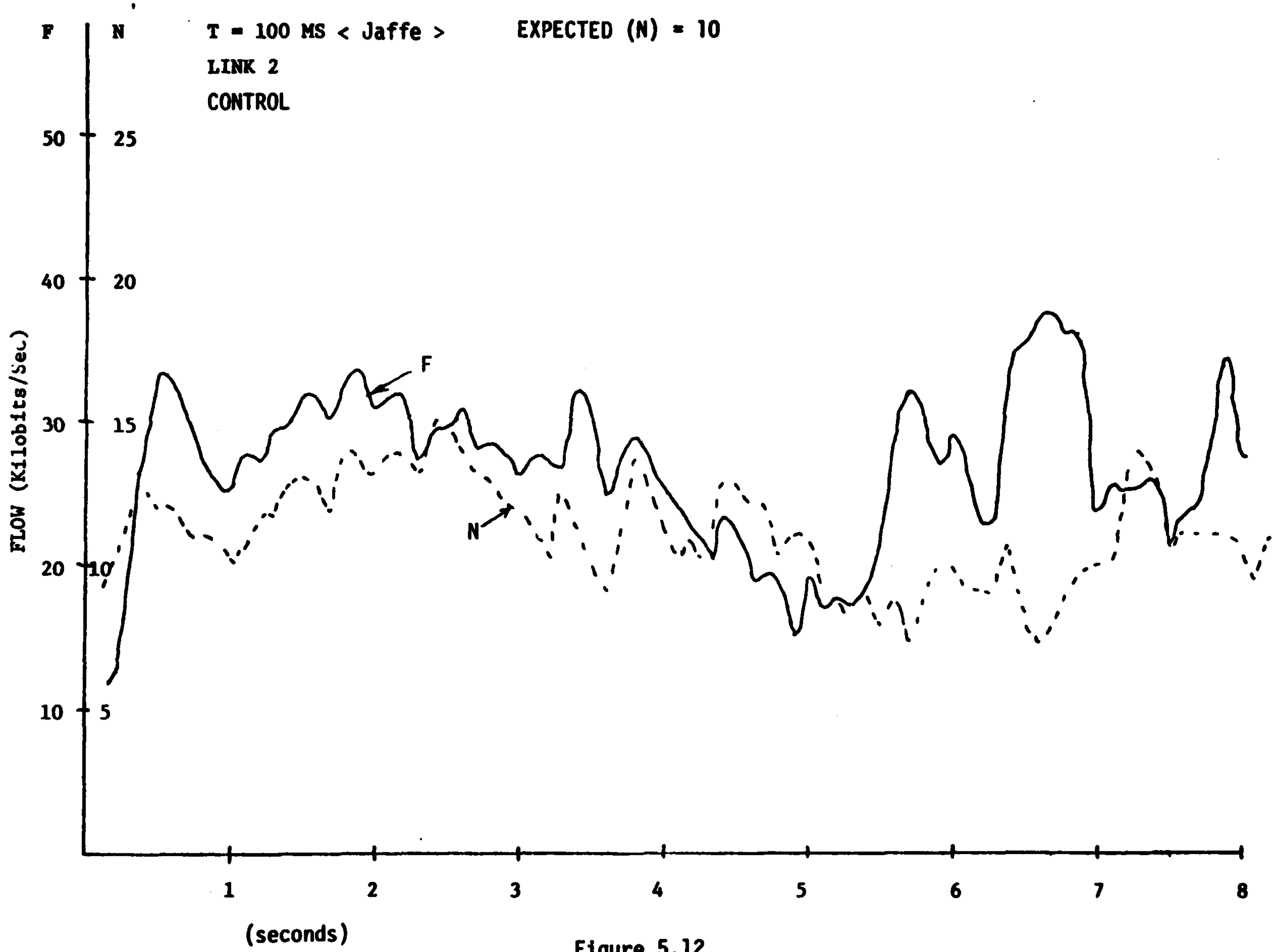

F
N
T = 100 MS < Jaffe >
EXPECTED (N) = 10
LINK 2
CONTROL
50
40
30
20
10
25
20
15
10
5
FLOW (Kilobits/Sec.)
F
N
1
2
3
4
5
6
7
8
(seconds)

Figure 5.12

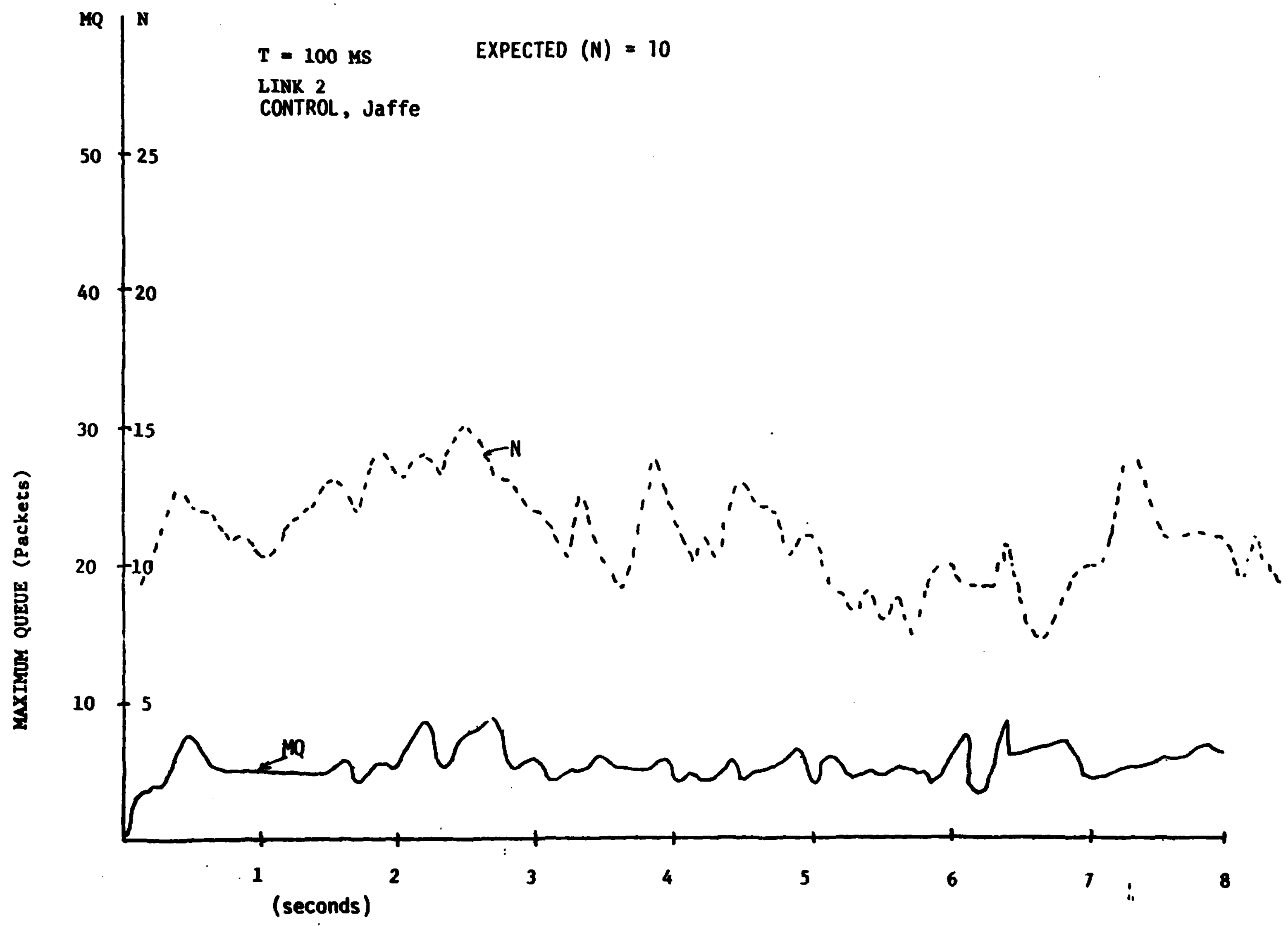

MQ
N
T = 100 MS
LINK 2
CONTROL, Jaffe
EXPECTED (N) = 10
MAXIMUM QUEUE (Packets)
50
40
30
20
10
25
20
15
10
5
N
MQ
1
2
3
4
5
6
7
8
(seconds)

Figure 5.13

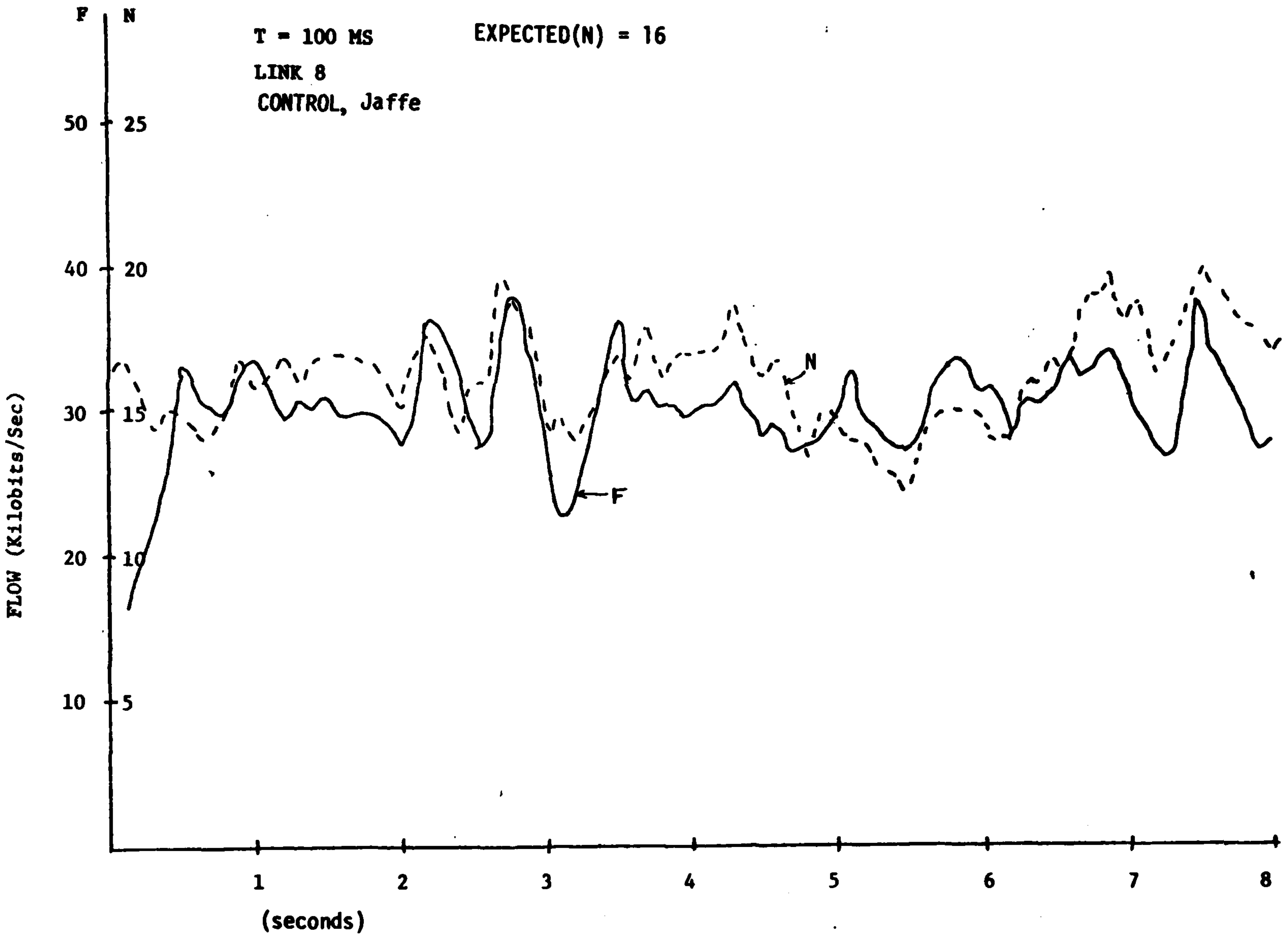

F
N
T = 100 MS
LINK 8
CONTROL, Jaffe
EXPECTED(N) = 16
50
25
40
20
30
15
20
10
10
5
FLOW (Kilobits/Sec)
1
2
3
4
5
6
7
8
(seconds)
F
N

Figure 5.14

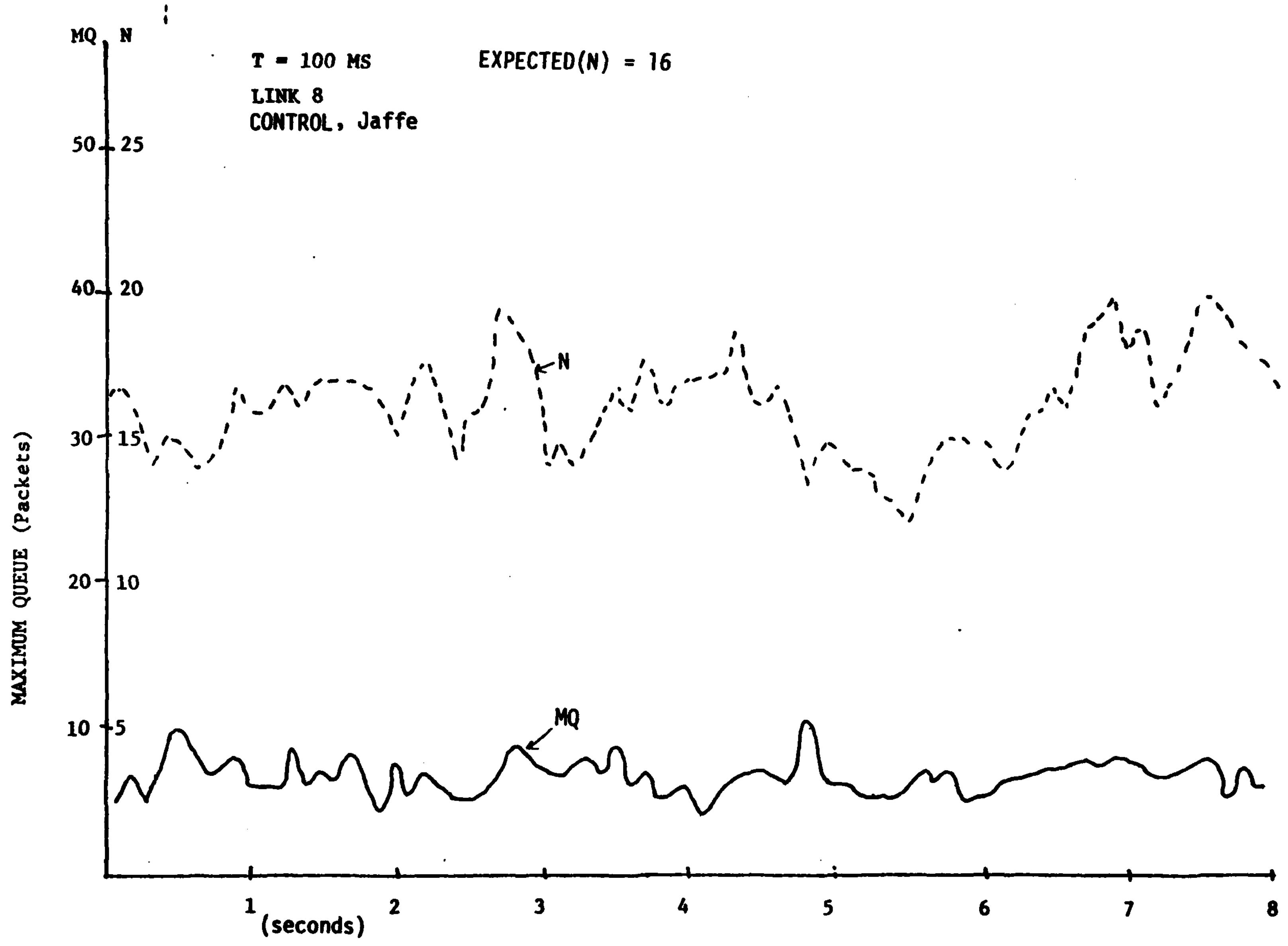

MQ, N
T = 100 MS
EXPECTED(N) = 16
LINK 8
CONTROL, Jaffe
50 25
40 20
30 15
20 10
10 5
MAXIMUM QUEUE (Packets)
N
MQ
1
2
3
4
5
6
7
8
(seconds)

Figure 5.15

D. Dynamic Source Rate Behavior

In Figure 5.16 we illustrate the dynamic behavior of sources 39 when our original algorithm is used. Since source 39 is controlled by link 2 we would expect its rate to increase when the flow on link 2 is below the effective capacity (i.e. 32 kb/s) and to decrease when $f_2 > c_2$, This can be verified by comparing Figure 5.16 with Figure 5.4.

The dynamic rate behavior of source 1 is shown in Figure 5.17.In this case the source rate is fairly constant due to the fact that the flow on link 8 (see Fig. 5.8) remains reasonably close to the effective capacity (i.e. 32 kb/s). It is also interesting to note the following; since link 8 controls all its users we expect (32/16) = 2 kb/s average rate/user and by examining Fig. 5.17 we see that this is indeed the case.

Similar results are shown in Figs. 5.18 - 5.19 for Jaffe's criteria. The only difference is that the height of the curve has been multiplied by a factor of $(\frac{10}{11})$ for source 39 and by $(\frac{16}{17})$ for source 1, to compensate for the additional excess capacity required by Jaffe's criteria.

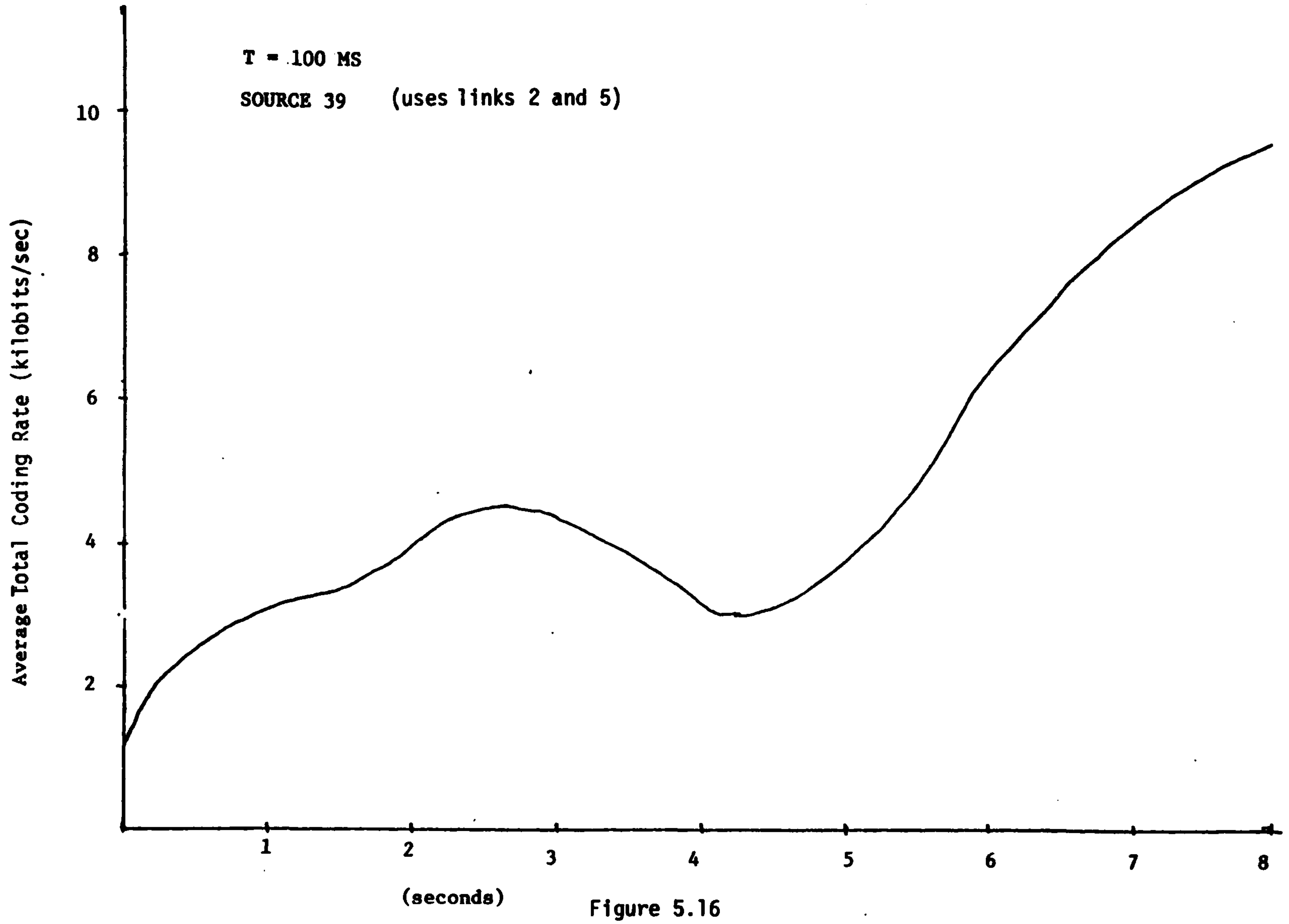

T = 100 MS
SOURCE 39 (uses links 2 and 5)
Average Total Coding Rate (kilobits/sec)
10
8
6
4
2
1
2
3
4
5
6
7
8
(seconds)

Figure 5.16

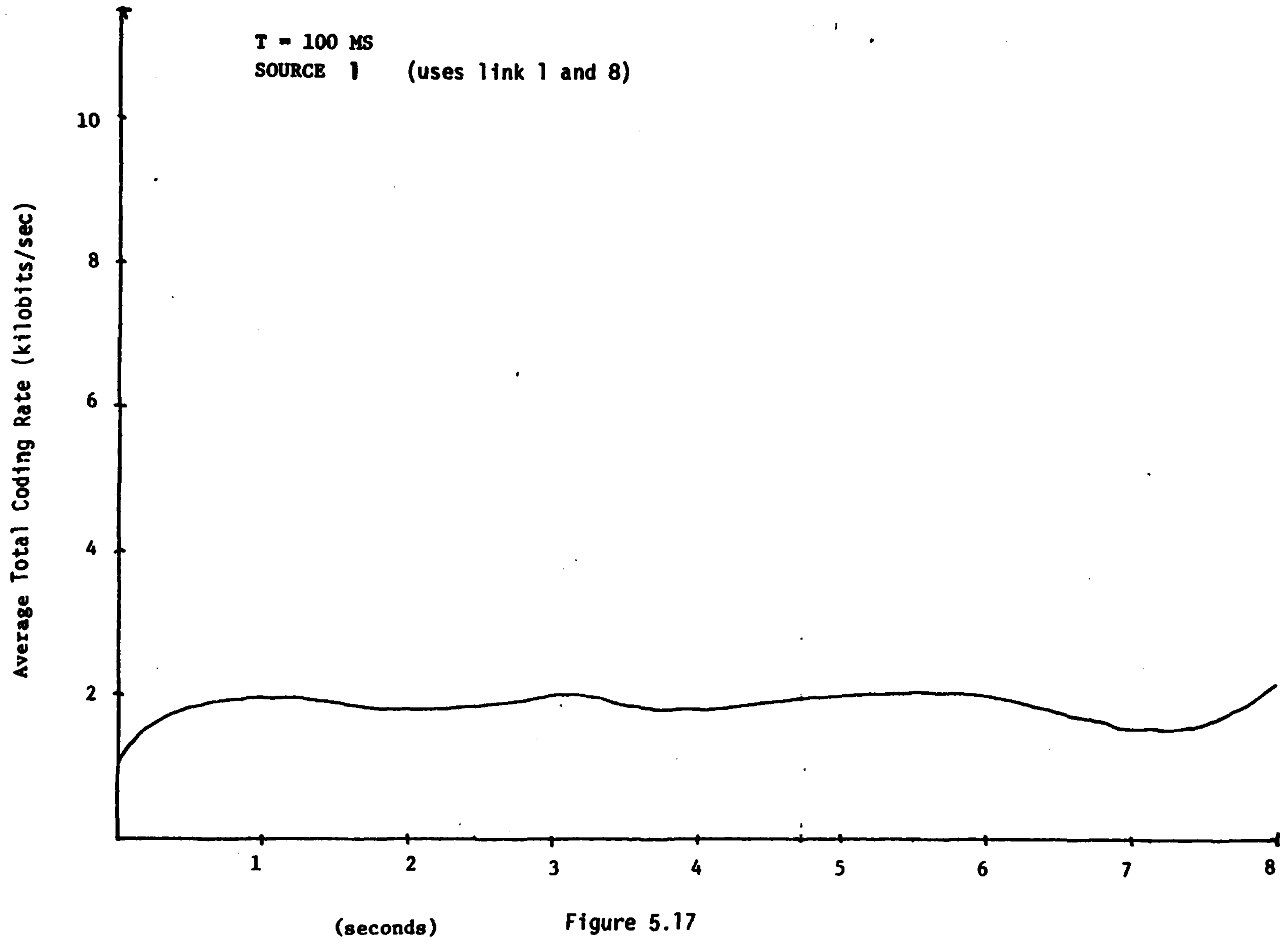

T = 100 MS
SOURCE 1 (uses link 1 and 8)
Average Total Coding Rate (kilobits/sec)
10
8
6
4
2
1
2
3
4
5
6
7
8
(seconds)

Figure 5.17

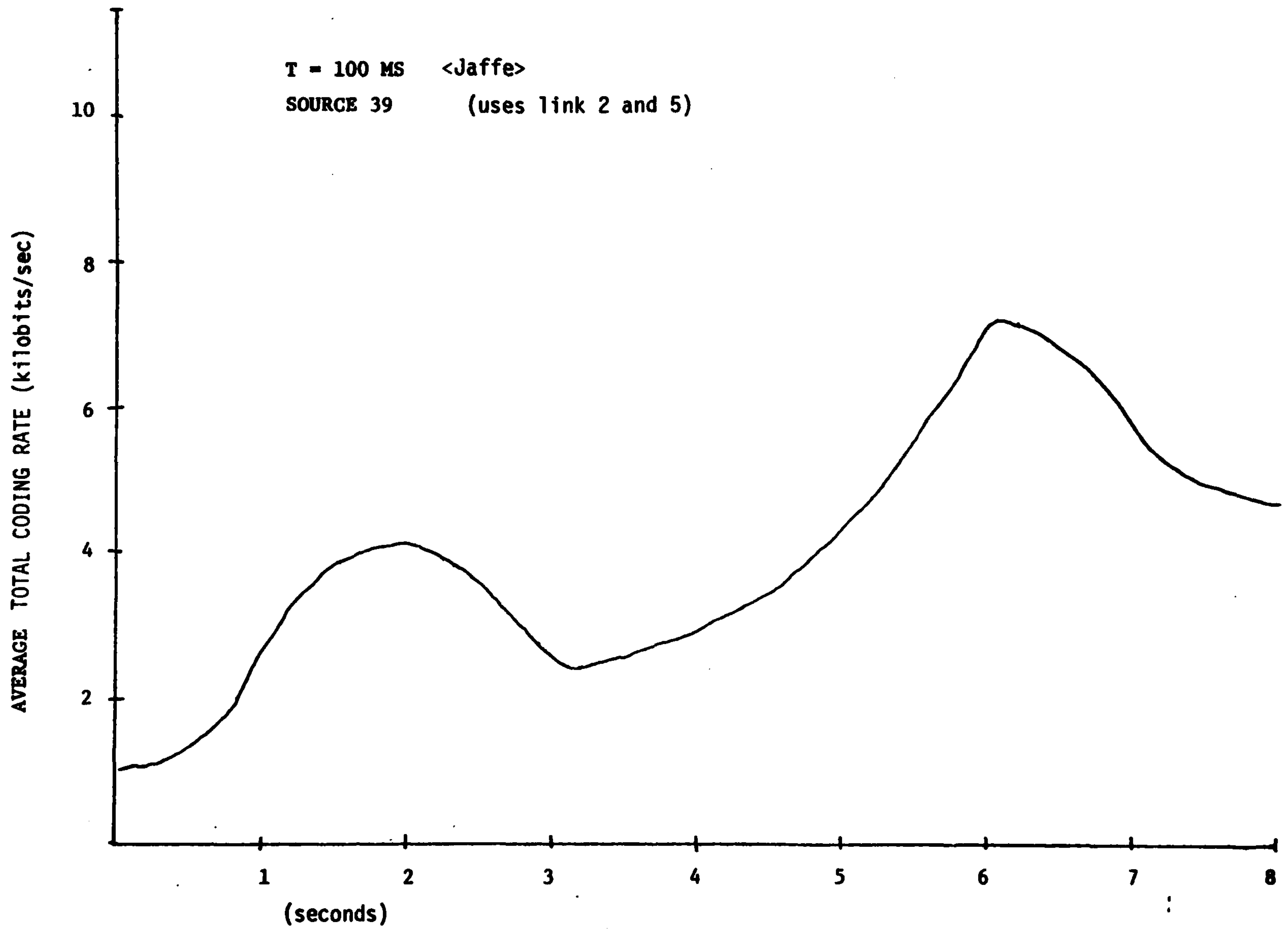

T = 100 MS <Jaffe>
SOURCE 39 (uses link 2 and 5)
AVERAGE TOTAL CODING RATE (kilobits/sec)
10
8
6
4
2
1
2
3
4
5
6
7
8
(seconds)

Figure 5.18

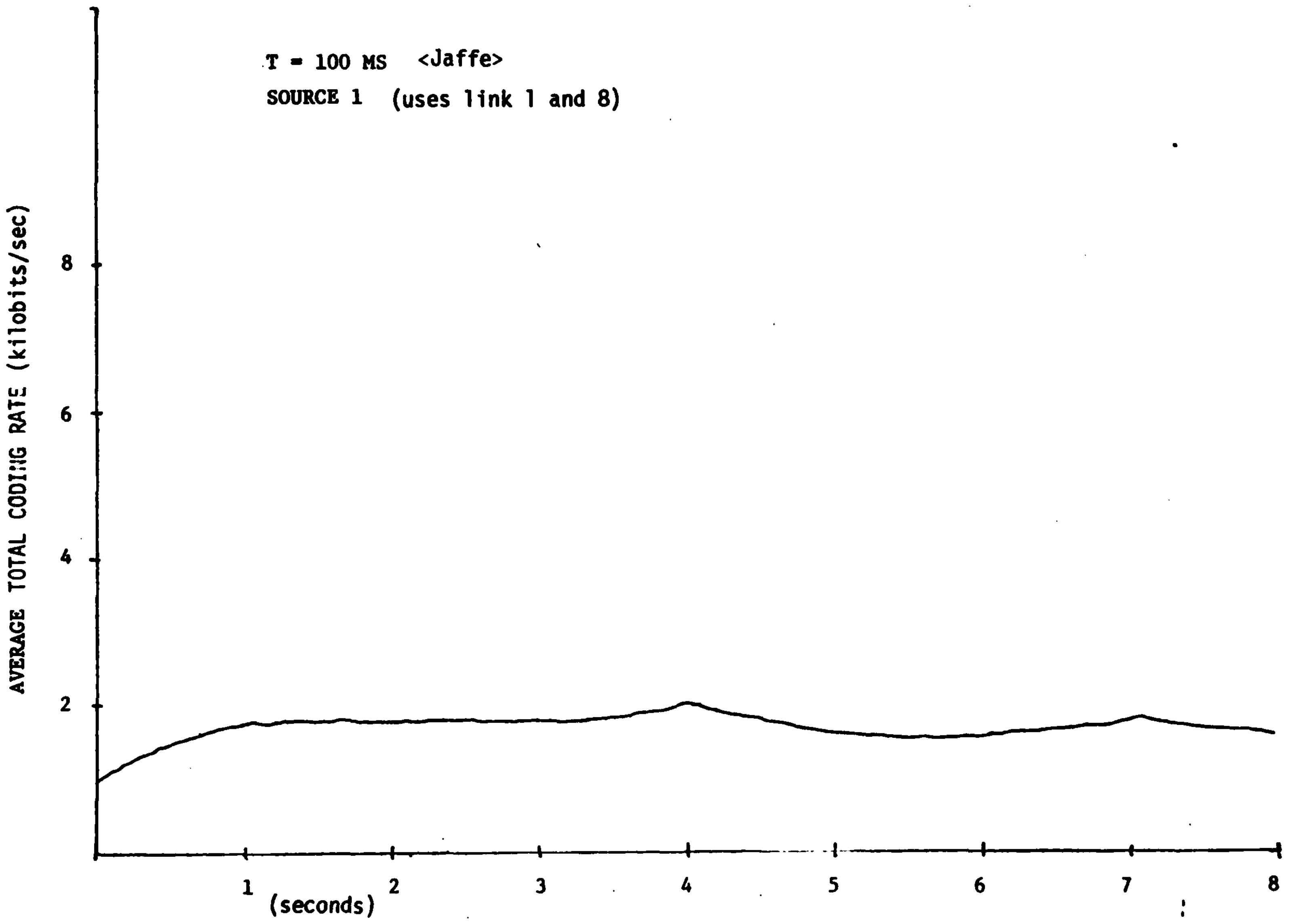

T = 100 MS <Jaffe>
SOURCE 1 (uses link 1 and 8)
AVERAGE TOTAL CODING RATE (kilobits/sec)
8
6
4
2
1
2
3
4
5
6
7
8
(seconds)

Figure 5.19

Statistics

Tables 5.1 - 5.6 summarize some key performance results for the three cases previously discussed. Several conclusions can be drawn and are as follows:

(1) No-control yields the worst performance in terms of delay. The straight control case yields reasonable delay characteristics with a bonus of in general higher coding rates when compared to the No-Control case. Incorporating Jaffe's idea results in superior delay characteristics at the expense of having the lowest coding rates.

(2) As expected from our previous discussion Jaffe's scheme yields superior performance in terms of MAX QUEUE size. However, this is again achieved at the expense of having the lowest link utilization values.

(3) An observation which can't be explained is the fact that in almost all cases the link utilization is below the desired value of 80%.

(4) It also should be noted that since links 1,2,3, and 4 have equivalent characteristics that they should yield equivalent performance especially in terms of link utilization. However, observing Fig. 5.4 - 5.6 we find this is indeed not the case. The reason for this being that the simulation runs were definitely not long enough to be able to compute averages although they yielded informative sample path behavior.

SOURCE - STATISTICS

T = 100 MS

AVERAGE PERIOD = 10 SECONDS

SOURCE	AVERAGE VOICE DELAY (10-4s)	MAX. VOICE DELAY (10-4s)	AVERAGE CONTROL DELAY (10-4s)	MAX. CONTROL DELAY (10-4s)	AVERAGE TOTAL CODING RATE (10^3 B/S)
1	275	31553	174	389	1.97
9	182	500	170	578	2.77
15	310	849	230	618	4.51
19	503	1308	238	1235	7.28
39	445	1227	269	1096	5.39

Table 5.1

SOURCE STATISTICS

NO CONTROL

AVERAGE PERIOD = 10 SECONDS

SOURCE	AVERAGE VOICE DELAY (10-4s)	MAX VOICE DELAY (10-4s)	AVERAGE CONTROL DELAY (10-4s)	MAX CONTROL DELAY (10-4s)	AVERAGE TOTAL CODING RATE (10^3 B/S)
1	345	926	120	170	2.0
9	303	942	194	808	2.67
15	465	2097	323	2089	4.0
19	435	933	276	1179	8.0
39	550	2254	467	1987	8.0

Table 5.2

SOURCE STATISTICS

T = 100 MS <J>

AVERAGE PERIOD = 10 SECONDS

SOURCE	AVERAGE VOICE DELAY (10-4s)	MAX VOICE DELAY (10-4s)	AVERAGE CONTROL DELAY (10-4s)	MAX CONTROL DELAY (10-4s)	AVERAGE TOTAL CODING RATE (10^3 B/S)
1	122	241	112	144	1.82
9	125	189	117	191	2.60
15	129	355	119	325	3.31
19	178	222	100	174	6.94
39	132	214	95	127	4.95

Table 5.3

LINK - STATISTICS

T = 100 MS

AVERAGE PERIOD = 10 SECONDS

LINK	AVERAGE FLOW (B/S)	UTILIZATION	AVERAGE QUEUE (PACKETS)	MAXIMUM QUEUE (PACKETS)
1	30543	76%	5	32
2	30707	77%	9	65
3	31418	79%	16	105
4	28340	70%	8	58
5	24230	61%	5	28
6	28846	72%	6	21
7	30276	76%	4	31
8	31170	78%	13	108

Table 5.4

LINK - STATISTICS

NO CONTROL

AVERAGE PERIOD = 10 SECONDS

LINK	AVERAGE FLOW (B/S)	UTILIZATION	AVERAGE QUEUE (PACKETS)	MAXIMUM QUEUE (PACKETS)
1	29167	73%	7	56
2	33560	84%	19	138
3	29791	74%	5	22
4	27805	70%	16	105
5	28530	71%	5	22
6	30367	76%	10	88
7	32640	82%	3	12
8	30575	86%	11	65

Table 5.5

LINK - STATISTICS

T = 100 MS <J>

AVERAGE PERIOD = 10 SECONDS

LINK	AVERAGE FLOW (B/S)	UTILIZATION	AVERAGE QUEUE (PACK)	MAX QUEUE (PACK)
1	24477	61%	1	7
2	27600	69%	2	9
3	24553	61%	2	8
4	24820	62%	2	11
5	15280	38%	1	4
6	26900	67%	1	17
7	29233	73%	3	8
8	30750	77%	7	19

Table 5.6

5.5 Comparison to the Lincoln Laboratory Scheme

Our simulation was relatively very detailed in that it monitored each packet generated by each source, whereas the L.L. simulation did not. Furthermore in [11] there are no results presented regarding packet delays, link queue sizes, or dynamic link behavior. The only results which we presented concern the dynamic behavior of two particular sources. Another important difference between the two simulations was that we were only concerned with (1/10) as many sources. Thus as one may expect, results regarding dynamic behavior would be more smoother in the L.L. case since they have the law of large numbers on their side.

Thus in conclusion, it is not possible for us to offer a fair comparison between our scheme and the Lincoln Laboratory scheme.

CHAPTER VI

CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

In this thesis we have presented two flow control algorithms. One algorithm was based on an optimization theoretic approach. The second algorithm had the property that the rate assignment for a particular source was dependent only upon the network condition and independent of the number of links which the source utilized. Thus the second algorithm appears more desirable for packet-voice networks primarily because of the proceeding property.

A computer simulation program was developed and used to examine the performance of the second algorithm and its extension. Results in general were good and coincided with our expectations. However our results indicated that future work is indeed necessary regarding two issues:

(1) Statistics clearly indicated that the duration of the simulation experiments were definitely not long enough to get accurate results.

(2) Our algorithmic development was based upon a quasi-static assumption, but after examining the various sample path behavior (Fig. 5.3 - 5.13) and statistics tables (Tables 5.1 - 5.6) it is clear that more work should be carried out to carefully study the dynamic behavior of the network traffic.

APPENDIX

Computer Simulation Program

```
%global static
c
c
c         time basis for simulation = 10-4 seconds
c
c
c
c
      parameter (n=8)
      parameter (m=80)
c
      dimension ivpcnt(m),icpcnt(m),mxdelv(m),mxdelc(m)
      dimension mxqcnt(n)
c
      integer cap(n),tprop(n),weight(n),tobs(n)
      integer contrl(n),qcount(n),qhead(n),qtail(n)
      integer bitcnt(n),tproc(5),packet(2000,9),pkhd
      integer etable(2000,7),conv(m,3),clock
      integer scount(m),scntrl(m),dcntrl(m)
      integer delayv(m),delayc(m)
      integer iqout(n),ibout(n),wtcnt(n)
      integer statl(8000,8),sprint(80),stats(8000,8)
c
      external random_$uniform(descriptors)
c
      common /stat3/iqout,ibout
      common /link1/cap,tprop
      common /link2/weight
      common /link3/tobs
      common /link4/contrl
      common /link5/qcount,qhead,qtail
      common /link6/bitcnt
      common /protm/tproc
      common /table1/packet,pkhd
      common /table2/etable,iearly,last,latest,ifptr
      common /table3/conv
      common /table4/statl,index
      common /table5/stats,jndex
      common /time/clock
      common /source/scount,scntrl,dcntrl
      common /gen/lenav,lencon,icpinc
      common /prnt/iprnt
      common /stat1/ivpcnt,icpcnt,mxdelv,mxdelc,delayv,delayc
      common /stat2/mxqcnt
      common /stat4/wtcnt
      common /psource/sprint
c                    .
c
c          set the simulation run duration
c
      data maxtim/110000/
c
c
c          set iprin=1 if wish to print initial data
c
      data iprin/0/
c
c
c ******************************************************************
c
c
c
c
c          initialize arrays and pointers
c
      do 5 i=1,8000
         do 6 j=1,8
            stat1(i,j) = 0
            stats(i,j) = 0
    6    continue
    5 continue
c
c
      do 10 i=1,2000
            etable(i,7) = 0
            packet(i;9) = 0
   10 continue
c
c
c
      ifptr  = 1
      iearly = 1
      last   = 1
      latest = 1
      pkhd   = 1
      index  = 0
      jndex  = 0
c
```

```
c
      do 7 i =1,n
           qcount(i) =0
           bitcnt(i) =0
           mxqcnt(i) =0
           iqout(i)  =0
           ibout(i)  =0
           wtcnt(i)  =0
    7 continue
c
      do 8 i =1,m
           ivpcnt(i) =0
           icpcnt(i) =0
           delayv(i) =0
           delayc(i) =0
           mxdelv(i) =0
           mxdelc(i) =0
           scount(i) =0
     8 continue
c
c
c
c          set up the network using the conversation matrix
c
      do 11 k=1,4
         do 12 j=1,8
            ival = j + 20*(k-1)
            conv(ival,1) = k
            conv(ival,2) = 8
            conv(ival,3) = -(81 - ival)
   12    continue
   11 continue
c
      do 13 k=1,4
         do 14 j=1,6
            ival = 8 + j + 20*(k-1)
            conv(ival,1) = k
            conv(ival,2) = 7
            conv(ival,3) = -(81 - ival)
   14    continue
   13 continue
c
      do 15 k =1,4
         do 16 j =1,4
```

```
            ival = 14 + j + 20*(k-1)
            conv(ival,1) = k
            conv(ival,2) = 6
            conv(ival,3) = -(81 - ival)
   16    continue
   15 continue
c
      do 17 k =1,4
         do 18 j =1,2
            ival = 18 + j + 20*(k-1)
            conv(ival,1) = k
            conv(ival,2) = 5
            conv(ival,3) = -(81 -ival)
   18    continue
   17 continue
c
c
c
          generate a random duration talkspurt for
          each initially active user
c
      do 1 j=2,80,2
         call random_$uniform(randn)
         arg = -lenav*log(randn)
         scount(j) = int(arg) + 1
c
c
c
c         determine the time of generation of the
          first talkspurt packet for each initially
          active source
c
         call random_$uniform(randt)
         istart = int(200*randt) + 1
         if(j.eq.2) istart = 0
         call event(0,-j,4,istart)
    1 continue
c
c
c         determine the time of generation of the first
          control packet for each initially silent
c         source
c
      do 2 j=1,79,2
```

```
        call random_$uniform(randt)
        istart = int(200*randt) + 1
        call event(0,j,4,istart)
    2 continue
c
c
c
c          set the initial scntrl and dcntrl values for
           each source
c
      do 3 k=1,m
         scntrl(k) = 500
         dcntrl(k) = 1000
    3 continue
c
c
c          set the initial control value for each
c          link and schedule the next link update.
c
      do 4 k=1,n
         contrl(k) = 1000
         call event(0,k,5,tobs(k))
    4 continue
c
c
c          schedule the first statistic computations
c
c       call event(0,0,6,5000)
c
c
c

c          determine the number of active speakers
c          initially on each link.
c
      do 30 i =1,m
         if(scount(i).eq.0) go to 30
         do 31 j =1,n
            jlink = conv(i,j)
            if(jlink.lt.0) go to 30
            wtcnt(jlink) = wtcnt(jlink) + 1
   31    continue
   30 continue
c
```

```
      do 33 i=1,m
          sprint(i) = 0
   33 continue
c
c
c         set sprint(i)=1 if wish statistics taken for
c         source i.
c
      do 34 k=1,4
            sprint(1 +(k-1)*20) = 1
            sprint(9 +(k-1)*20) = 1
            sprint(15+(k-1)*20) = 1
            sprint(19+(k-1)*20) = 1
   34 continue
c
c -------------------------------------------------------------------
c
c
c         print initial data if desired.
c
      if(iprin.eq.0) go to 19
c
c
      write(6,40)
   40 format(1h1,20x,'***************   initial data for run  ******************')
c
      do 49 i =1,n
             write(6,51) i
             write(6,52) cap(i)
             write(6,61) weight(i)
             write(6,53) tprop(i)
             write(6,54) contrl(i)
             write(6,55) wtcnt(i)
c
   51        format('0',10x,'link number =',2x,i5)
   52        format(10x,'capacity =',2x,i8,2x,'bits/sec')
   61        format(10x,'weight   =',2x,i5)
   53        format(10x,'propagation delay =',2x,i4,2x,'10-4sec')
   54        format(10x,'initial control value =',2x,i8,2x,'bits/sec')
   55        format(10x,'initial number of active talkers =',2x,i6)
   49 continue
c
       do 70 j =1,m
              write (6,71)j
```

```
                 write (6,72) scntrl(j)
                 write (6,73) dcntrl(j)
                 write (6,74) scount(j)
c
   71            format('0',10x,'source number=',2x,i4)
   72            format(10x,'coding rate    =',2x,i8,2x,'bits/sec')
   73            format(10x,'feedback rate =',2x,i8,2x,'bits/sec')
   74            format(10x,'initial number of voice packets =',i6)
   70 continue
c
      write(6,60)
   60 format('0',20x,'event processing time')
      write(6,21) tproc(1)
   21 format(10x,'absorbtion     =',2x,i4,2x,'10-4sec')
      write(6,22) tproc(2)
   22 format(10x,'arrival        =',2x,i4,2x,'10-4sec')
      write(6,23) tproc(3)
   23 format(10x,'transmission   =',2x,i4,2x,'10-4sec')
      write(6,24) tproc(4)
   24 format(10x,'generation     =',2x,i4,2x,'10-4sec')
      write(6,25) tproc(5)
   25 format(10x,'control update =',2x,i4,2x,'10-4sec')
c
       write(6,91) icpinc
   91 format('0',10x,'control packet intergeneration time interval =',2x,i4,2x,'10-4sec')
c
      write(6,92) lencon
   92 format('0',10x,'overhead number of bits in packet       =',2x,i4)
      write(6,94) lenav
   94 format('0',10x,'average number of packets per talkspurt =',2x,i4)
c
c
c ---------------------------------------------------------------------------
    19 write(6,20) maxtim
    20 format(1h1,20x,'simulation run time duration =',2x,i8,2x,'10-4seconds')
c
c
c
```

```
c
c
c       ***********************************************
c       *                                             *
c       *                                             *
c       *              MAIN  PROGRAM                  *
c       *                                             *
c       *                                             *
c       ***********************************************
c
c
c
c       set clock equal to time of current event
c
  50 clock = etable(iearly,1)
c
c
c
c       call proper subroutine to carry out current event
c
c
c
c
     id =  etable(iearly,3)
     loc = etable(iearly,4)
c
c
c
     go to (100,200,300,400,500,600)   etable(iearly,2)
c
c
 100 call absorb(id,loc)
     go to 1000
 200 call arrive(id,loc)
     go to 1000
 300 call xmit(id,loc)
     go to 1000
 400 call sgen(loc)
     go to 1000
 500 call update(loc)
     go to 1000
 600 call stat
c
c
```

```
c
 1000 etable(iearly,7) =0
c
c
c           determine the next event
c
        iearly = etable(iearly,5)
c
c
        if(clock.lt.maxtim) go to 50
c
c
      call print
c
      stop
      end
c
c
```

```
c
c
c
c     ****************************
c     *                          *
c     *      BLOCK   DATA        *
c     *                          *
c     *                          *
c     ****************************
      block data
c
      parameter (n=8)
      parameter (m=80)
c
      integer cap(n),tprop(n),weight(n),tobs(n)
      integer tproc(5),conv(m,3)
c
      common /link1/cap,tprop
      common /link2/weight
      common /link3/tobs
      common /protm/tproc
      common /table3/conv
      common /gen/lenav,lencon,icpinc
c
c
c
c                 set the network characteristics
c
      data cap(1),cap(2),cap(3),cap(4)/40000,40000,40000,40000/
      data cap(5),cap(6),cap(7),cap(8)/40000,40000,40000,40000/
      data tprop(1),tprop(2),tprop(3),tprop(4)/30,30,30,30/
      data tprop(5),tprop(6),tprop(7),tprop(8)/30,30,30,30/
      data weight(1),weight(2),weight(3),weight(4)/20,20,20,20/
      data weight(5),weight(6),weight(7),weight(8)/8,16,24,32/
      data tobs(1),tobs(2),tobs(3),tobs(4)/1000,1000,1000,1000/
      data tobs(5),tobs(6),tobs(7),tobs(8)/1000,1000,1000,1000/
      data tproc(1),tproc(2),tproc(3),tproc(4),tproc(5)/5,5,1,5,5/
      data lenav,lencon/60,10/
      data icpinc/1000/
c
c
c
      end
```

```
c         ****************************************************
c         *                                                  *
          *               SUBROUTINE  EVENT                  *
c         *                                                  *
          ****************************************************
c
c
c                   subout ine event - schedules a future event.
c                             each time an event is added to
c                             the EVENT table, the table must
                              be resorted as to maintain
c                             chronological order.
c
c
c
c
      subroutine event(id,loc,itype,itime)
c
c
      integer etable(2000,7)
c
      common/table2/etable,iearly,last,latest,ifptr
c
  100 if(etable(ifptr,7).ne.1) go to 120
      ifptr = ifptr +1
      if(ifptr.eq.2001) ifptr = 1
      go to 100
  120 itemp = ifptr
c
c
c
c
      etable(itemp,1) = itime
      etable(itemp,2) = itype
      etable(itemp,3) = id
      etable(itemp,4) = loc
      etable(itemp,7) = 1
c
c
      if(itime.ge.etable(last,1))    go to 10
      if(itime.ge.etable(latest,1))  go to 20
c
      jrow = etable(latest,6)
    1 if(etable(jrow,1).le.itime)   go to 30
```

```
      jrow = etable(jrow,6)
      go to 1
c
   20 jrow = etable(latest,5)
    2 if(etable(jrow,1).ge.itime) go to 40
      jrow = etable(jrow,5)
      go to 2
c
   30 irow = etable(jrow,5)
      etable(itemp,5) = irow
      etable(itemp,6) = jrow
      etable(irow,6) = itemp
      etable(jrow,5)  = itemp
      go to 50
c
   40 irow = etable(jrow,6)
      etable(itemp,6) = irow
      etable(itemp,5) = jrow
      etable(irow,5)  = itemp
      etable(jrow,6)  = itemp
      go to 50
c
   10 etable(itemp,6) = last
      etable(itemp,5) = 0
      etable(last,5)  = itemp
      last = itemp
c
   50 latest = itemp
      ifptr = ifptr + 1
      if(ifptr.eq.2001) ifptr = 1
      return
      end
c
c
```

```
c        ****************************************
c        *                                      *
c        *         SUBROUTINE  ABSORB           *
c        *                                      *
c        *                                      *
c        ****************************************
c
c
c                  subroutine absorb handles the
c                           arrival of a packet
c                           at its destination.
c
c
c
c
      subroutine absorb(id,loc)
c
      parameter (m=80)
      parameter (n=8)
c
      dimension ivpcnt(m),icpcnt(m),mxdelv(m),mxdelc(m)
c
c
      integer delay,packet(2000,9),pkhd,tproc(5)
      integer scntrl(m),dcntrl(m),scount(m),clock
      integer conv(m,3)
      integer delayv(m),delayc(m)
      integer etable(2000,7)
      integer wtcnt(n)
c
      common /table1/packet,pkhd
      common /table2/etable,iearly,last,latest,ifptr
      common /table3/conv
      common /source/scount,scntrl,dcntrl
      common /gen/lenav,lencon,icpinc
      common /protm/tproc
      common /time/clock
      common /stat1/ivpcnt,icpcnt,mxdelv,mxdelc,delayv,delayc
      common /stat4/wtcnt
c
c
      itime = clock + tproc(1)
c
```

```
c
c                   compute delay of packet
c
      delay = itime - packet(id,7)
c
c
c
c                   update delay statistics for source
c
      nums = packet(id,1)
      if(packet(id,6).eq.2) go to 40
c
      delayv(nums) = delayv(nums) + delay
      ivpcnt(nums) = ivpcnt(nums) + 1
      mxdelv(nums) = max(delay,mxdelv(nums))
      go to 45
c
   40 delayc(nums) = delayc(nums) + delay
      icpcnt(nums) = icpcnt(nums) + 1
      mxdelc(nums) = max(delay,mxdelc(nums))
c
c
c
c                   update control values for source
c
   45 dcntrl(loc) = packet(id,4)
      scntrl(loc) = packet(id,5)
c
c
c                   eliminate presence of packet from network
c
      packet(id,9) = 0
c
c ------------------------------------------------------------
c
c                   determine if last voice packet in
c                            current talkspurt, if not
c                            then return, else continue
c
c
      if(packet(id,6).ne.1) return
c
c
c
```

```
c
c
c                 generate new talkspurt
c
      external random_$uniform(descriptors)
      call random_$uniform(unif)
      arg =  -lenav*log(unif)
      number = int(arg) +1
      scount(loc) = number
c
c
c                 update the number of active speakers
c                         on each link
c
      isend = packet(id,1)
      do 200 j = 1,n
            jlink = conv(isend,j)
            if (jlink.lt.0) go to 201
            wtcnt(jlink) = wtcnt(jlink) - 1
  200 continue
c
  201 do 202 j =1,n
            jlink = conv(loc,j)
            if(jlink.lt.0) go to 203
            wtcnt(jlink) = wtcnt(jlink) + 1
  202 continue
c
c
c
c                 schedule generation of first talkspurt packet
c
c
  203 call event(0,-loc,4,itime+1)
c
c
c
c
     return
      end
c
c
```

```
c
c      *************************************
c      *                                   *
       *        SUBROUTINE  ARRIVE         *
c      *                                   *
c      *************************************
c
c
c
c                subroutine arrive - handles
                           the arrival of a
                           packet at a link
c
c
c
c
      subroutine arrive(id,link)
c
      parameter(n=8)
c
      dimension mxqcnt(n)
c
      integer contrl(n),qcount(n),qhead(n),qtail(n)
      integer bitcnt(n),tproc(5),etable(2000,7)
      integer packet(2000,9),clock,pkhd
c
      common /link4/contrl
      common /link5/qcount,qhead,qtail
      common /link6/bitcnt
      common /protm/tproc
      common /table1/packet,pkhd
      common /table2/etable,learly,last,latest,ifptr
      common /time/clock
      common /stat2/mxqcnt
c
c
c
c
c                update link statistics
c
c
      qcount(link) = qcount(link) +1
      bitcnt(link) = bitcnt(link) + packet(id,3)
      mxqcnt(link) = max(qcount(link),mxqcnt(link))
c
c
c
c
                       if queue was empty before arrival
c                      then schedule transmission of packet
c                      if not, place packet at end of queue.
c
c
      if(qcount(link).eq.1) go to 10
c
c
      packet(qtail(link),8) = id
      qtail(link) = id
      return
c
c
   10 itime =clock + tproc(2)
      qtail(link) = id
      call event(id,link,3,itime)
c
      return
      end
c
```

```
c
c      ***********************************
c      *                                 *
       *                                 *
c      *                                 *
c      *      SUBROUTINE  XMIT           *
c      *                                 *
c      *                                 *
       ***********************************
c
c
c
c
c                 subroutine transmit- handles
                            the transmission of
                            a packet on a link
c
c
c
      subroutine xmit(id,link)
c
      parameter(n=8)
      parameter (m=80)
c
      integer qcount(n),qhead(n),qtail(n),bitcnt(n)
      integer tproc(5),packet(2000,9),pkhd
      integer conv(m,3),clock,etable(2000,7)
      integer cap(n),tprop(n)
      integer contrl(n)
      integer iqout(n),ibout(n)
c
      common /link1/cap,tprop
      common /link4/contrl
      common /link5/qcount,qhead,qtail
      common /link6/bitcnt
      common /protm/tproc
      common /table1/packet,pkhd
      common /table2/etable,iearly,last,latest,ifptr
      common /table3/conv
      common /time/clock
      common /stat3/iqout, ibout
c
c
      if(id.gt.0) go to 10
```

```
c
c
      qcount(link) = qcount(link) - 1
      if(qcount(link).ne.0)  go to 9
      return
c
    9 id = packet(-id,8)
   10 iflag = 2
c
c
c                     determine the time of packet arrival
                      at the next location.
c
c
      itrans =(packet(id,3)*10000)/cap(link)
      itime = clock + itrans + tproc(3)
c
c
c                     determine the next location for the packet
c
      packet(id,2) = packet(id,2) +1
      nloc = conv(packet(id,1),packet(id,2))
c
c
c
c                     if the next location is the receiver
c                     then schedule an absorbtion.
c
      if(nloc.gt.0)  go to 300
      iflag = 1
      nloc = -nloc
c
c -----------------------------------------------------------
c
c
c                     update the feedforward control field
c                     using the latest link control value
c
  300 packet(id,4) = min (packet(id,4),contrl(link))
c
c
c
c                     update link output statistics
c
```

```
      iqout(link) = iqout(link) + 1
      ibout(link) = ibout(link) + packet(id,3)
c
c
c                schedule packet at next location
c
      call event(id,nloc,iflag,itime + tprop(link))
c
c
c                schedule the deletion of the current
c                packet being serviced.
c
      call event(-id,link,3,itime)
c
      return
      end
c
```

```
c
c        ********************************************
c        *                                          *
         *                                          *
c        *           SUBROUTINE  SGEN               *
c        *                                          *
c        *                                          *
c        ********************************************
c
c
c
c
c                    subroutine sgen - handles the
c                             generation of a new
c                             packet from a source.
c
c
      subroutine sgen(node)
c
c
      parameter(m=80)
c
      integer scount(m),scntrl(m),dcntrl(m)
      integer packet(2000,9),tproc(5),clock
      integer etable(2000,7),pkhd
      integer conv(m,3)
c
      common /table1/packet,pkhd
      common /protm/tproc
      common /table2/etable,iearly,last,latest,ifptr
      common /time/clock
      common /source/scount,scntrl,dcntrl
      common /gen/lenav,lencon,icpinc
      common /table3/conv
c
      data iperiod,irange/180,40/
c
c
      inode = abs(node)
      if(scount(inode).gt.0.and.node.gt.0) return
c
      node = abs(node)
c
c
```

```
    1 if(packet(pkhd,9).ne.1) go to 2
      pkhd = pkhd + 1
      if(pkhd.eq.2001) pkhd =1
      go to 1
c
c
c                create the packet
c
    2 packet(pkhd,9) = 1
      itype = 0
      if(scount(node).eq.0) go to 10
      if(scount(node).eq.1) itype =1
      scount(node) = scount(node) -1
      len = (scntrl(node)/50)
      go to 20
c
c
c
   10 len = lencon
      itype = 2
c
   20 packet(pkhd,1) = node
      packet(pkhd,2) = 1
      packet(pkhd,3) = len
      packet(pkhd,4) = 1000000
      packet(pkhd,5) = dcntrl(node)
      packet(pkhd,6) = itype
      packet(pkhd,7) = clock
      packet(pkhd,8)  =0
c
c ------------------------------------------------------------
c
      itime = clock + tproc(4)
c
c
c                schedule arrival of the generated packet
c                           at its first location.
c
      call event(pkhd,conv(node,1),2,itime)
c
c
c                if the packet is a voice packet and not
c                           the last voice packet of the
c                           current talkspurt then determine
```

```
c                                 the  time of generation of
c                                 the next voice packet.
c
      if (itype.eq.2) go to 200
c
c
      external random_$uniform(descriptors)
      call random_$uniform(rand)
      itinc = iperiod + (irange*rand)
c
      node = -node
      go to 210
c
c
c                       determine the time of generation of
c                                 the next control packet.
c
  200 itinc = icpinc
c
  210 itime = itime + itinc
c
c
c
c                       schedule the generation of the next packet
c
      call event(0,node,4,itime)
c
      pkhd = pkhd + 1
      if(pkhd.eq.2001) pkhd = 1
      return
      end
c
c
```

```
c        ***********************************
c        *                                 *
c        *                                 *
c        *    SUBROUTINE    UPDATE         *
c        *                                 *
c        *                                 *
c        ***********************************
c
c
c
c                  subroutine update - performs
c                             the update of the
c                             link control value.
c
c
c      subroutine update(link)
c
c
      parameter (n=8)
c
      integer cap(n),tprop(n),weight(n),tobs(n)
      integer contrl(n),tproc(5),etable(2000,7)
      integer bitcnt(n),clock
      integer qcount(n),qhead(n),qtail(n)
      integer iqout(n),ibout(n),wtcnt(n)
      integer statl(8000,8)
c
      dimension mxqcnt(n)
c
      common /link1/cap,tprop
      common /link2/weight
      common /link3/tobs
      common /link4/contrl
      common /link5/qcount,qhead,qtail
      common /link6/bitcnt
      common /protm/tproc
      common /table2/etable,iearly,last,latest,ifptr
      common /table4/statl,index
      common /time/clock
      common /prnt/iprnt
      common /stat2/mxqcnt
      common /stat3/iqout,ibout
      common /stat4/wtcnt
c
```

```
c
c
c ----------------------------------------------------------------------------
c
c
c
c                   compute the average flow over the
c                            previous observation period.
c
  300 iflow = (bitcnt(link)*10000)/tobs(link)
      iefcap = cap(link)*0.8
c
c
c                   compute the new link control value using
c                            the appropriate update equation.
c
c
      contrl(link)= max( min(contrl(link) + (iefcap-iflow)/weight(link), iefcap),iefcap/weight(link))
c
c ----------------------------------------------------------------------------
c
c
c
      index = index + 1
c
  310 statl(index,1) = link
      statl(index,2) = clock
      statl(index,3) = iflow
      statl(index,4) = contrl(link)
      statl(index,5) = wtcnt(link)
      statl(index,6) = mxqcnt(link)
      statl(index,7) = qcount(link)
      statl(index,8) = ibout(link)
c
c
c
c
c                   reset the link statistic monitors.
c
  320 bitcnt(link) = 0
      iqout(link)  = 0
c
c
      ibout(link)  = 0
      mxqcnt(link) = 0
```

```
c
c
c
c                    schedule the next link update.
c
c
      itime = clock + tobs(link)
c
c
      call event(0,link,5,itime)
c
      return
      end
c
c
c
```

```
c          ************************************
c          *                                  *
c          *                                  *
c          *        SUBROUTINE  STAT          *
c          *                                  *
c          ************************************
c
c
c
c                   subroutine stat - computes
c                       statistics of the sources.
c
c
c
      subroutine stat
c
      parameter (n=8)
      parameter (m=80)
c
      dimension ivpcnt(m),icpcnt(m),mxdelv(m),mxdelc(m)
      dimension mxqcnt(n)
      dimension iavdelv(m),iavdelc(m)
c
      integer delayv(m),delayc(m),clock,etable(2000,7)
      integer scount(m),scntrl(m),dcntrl(m)
      integer sprint(80),stats(8000,8)
c
      common /time/clock
      common /stat1/ivpcnt,icpcnt,mxdelv,mxdelc,delayv,delayc
      common /stat2/mxqcnt
      common /table2/etable,iearly,last,latest,ifptr
      common /table5/stats,jndex
      common /source/scount,scntrl,dcntrl
      common /psource/sprint
c
c
c
c                   set the period between statistic
c                             computations.
c
      data iperiod/5000/
c
c
c
```

```
c
c                    compute average delay statistics
c                            for each source
c
c      do 10 i=1,m
             if(ivpcnt(i).ne.0) go to 15
             iavdelv(i) = 0
             go to 16
   15        iavdelv(i) = delayv(i)/ivpcnt(i)
   16        if(icpcnt(i).ne.0) go to 17
             iavdelc(i) = 0
             go to 10
   17        iavdelc(i) = delayc(i)/icpcnt(i)
   10 continue
c
c
c -------------------------------------------------------------------
      do 120 j =1,m
             if(sprint(j).ne.1) go to 120
             jndex = jndex + 1
             stats(jndex,1) = j
             stats(jndex,2) = clock
             stats(jndex,3) = iavdelv(j)
             stats(jndex,4) = mxdelv(j)
             stats(jndex,5) = iavdelc(j)
             stats(jndex,6) = mxdelc(j)
             stats(jndex,7) = scntrl(j)
             stats(jndex,8) = dcntrl(j)
  120 continue
c
c -------------------------------------------------------------------
c
c
c
c
c                    reinitialize statistic arrays to zero
c
c
      do 200 j =1,m
             ivpcnt(j) =0
             icpcnt(j) =0
             mxdelv(j) =0
             mxdelc(j) =0
             iavdelv(j)=0
```

```
               iavdelc(j)=0
               delayv(j) =0
               delayc(j) =0
  200  continue
c
c
c
c
c
c          schedule the next statistics computation
c
      call event(0,0,6,clock+iperiod)
c
c
c
      return
      end
c
```

```
c
c
c          *******************************
c          *                             *
c          *      SUBROUTINE  PRINT      *
c          *                             *
c          *******************************
c
c
c                      subroutine print-
                       prints the link
c                      statistics.
c
c
      subroutine print
c
      integer statl(8000,8),stats(8000,8)
c
      common /table4/statl,index
      common /table5/stats,jndex
c
      write(6,1)
      write(6,2)
c
      do 5 i=1,index
         iflag = iflag + 1
         if(mod(iflag,49).ne.0) go to 6
         iflag = 0
         write(6,1)
         write(6,2)
    6  write(6,3)(statl(i,k),k=1,8)
    5 continue
c
      write(6,7)
      write(6,8)
c
      do 10 j = 1,jndex
           jflag = jflag + 1
           if(mod(jflag,49).ne.0) go to 12
           write(6,7)
           write(6,8)
           jflag = 0
   12      write(6,9)(stats(j,k),k=1,8)
   10 continue
```

```
c
c
    1 format(1h1,1x,'   link     clock      flow      new control    active    max-queue    queue   bits-xmitted')
    2 format(3x,'            (10-4s)      (b/s)        (b/s)                      (packets)    (pack)',///)
    3 format(3x,i4,4x,i8,4x,i8,4x,i8,6x,i4,8x,i4,7x,i4,6x,i8)
    7 format(1h1,1x,'source  clock   av.voice delay  max delay   av.contl delay  max delay   coding rate   feedback rate')
    8 format(1x,'        (10-4s)      (10-4s)          (10-4s)      (10-4s)          (10-4s)      (bits/sec)      (bits/sec)',///)
    9 format(2x,i4,3x,i6,6x,i5,10x,i5,7x,i5,10x,i5,6x,i7,8x,i7)
c
      return
      end
```

References

[1] K. Bullington and J.M. Fraser, "Engineering Aspects of TASI," Bell System Technical Journal, Vol. 38, March 1959.

[2] S.J. Campenella, "Digital Speech Interpolation," COMSAT Technical Review, Vol. 6, Spring 1976.

[3] J.A. Sciulli and S.J. Campenella, "A speech Predictive Encoding Communication System for Multichannel Telephony," IEEE Trans. Commun., Vol. COM-21, July 1973.

[4] B. Gold, "Digital Speech Networks," Proc. IEEE, Vol. 65, Dec. 1977.

[5] H. Frank and I. Gitman, "Economic Analysis of Integrated Voice and Data Networks; A Case Study," Proc. IEEE, Vol. 66, Nov. 1978.

[6] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," IEEE Trans. Commun., Vol. COM-28, April 1980.

[7] T. Bially, A.J. McLoughlin and C.J. Weistein, "Voice Communication in Integrated Digital Voice and Data Networks," IEEE Trans. Commun., Vol. COM-28, Sept. 1980.

[8] G. Karog, L. Fransen and E. Kline, "Multirate Processor (MRP)," Naval Research Laboratory Report, Sept. 1978.

[9] B. Gold, "Multiple Rate Channel Vocoding," EASCON 1978 Conference Record.

[10] T. Bially, B. Gold and S. Seneff, "A Technique for Adaptive Voice Flow Control in Integrated Packet Networks," IEEE Trans. Commun., Vol. COM-28, March 1980.

[11] S. Seneff, "Computer Simulation Model for a Digital Communications Network Utilizing an Embedded Speech Coding Technique," M.I.T. Lincoln Laboratory Technical Note, TN - 1978 - 33, Oct. 1978.

[12] D.G. Luenberger, Introduction to Linear and Nonlinear Programming, Addison - Wesley Publishing Company, Inc., Reading, Mass. 1973.

[13] G. Strang, Linear Algebra and its Applications, Academic Press, Inc. New York, 1976.

[14] J.M. Jaffe, "A Decentralized 'Optimal' Multiple-User Flow Control Algorithm," ICCC 1980 Conference Record.

[15] S.J.Golestaani, "A Unified Theory of Flow Control and Routing in Data Communication Networks", LIDS Report No. TH-963, M.I.T.,1980.

Distribution List

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 Copies
Assistant Chief for Technology Office of Naval Research, Code 200 Arlington, Virginia 22217	1 Copy
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 Copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 Copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 Copy
Office of Naval Research Branch Office, Pasadena 1030 East Greet Street Pasadena, California 91106	1 Copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 Copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 Copy

Office of Naval Research Code 455 Arlington, Virginia 22217	1 Copy
Office of Naval Research Code 458 Arlington, Virginia 22217	1 Copy
Naval Electronics Laboratory Center Advanced Software Technology Division Code 5200 San Diego, California 92152	1 Copy
Mr. E. H. Gleissner Naval Ship Research & Development Center Computation and Mathematics Department Bethesda, Maryland 20084	1 Copy
Captain Grace M. Hopper NAICOM/MIS Planning Branch (OP-916D) Office of Chief of Naval Operations Washington, D.C. 20350	1 Copy
Advanced Research Projects Agency Information Processing Techniques 1400 Wilson Boulevard Arlington, Virginia 22209	1 Copy
Dr. Stuart L. Brodsky Office of Naval Research Code 432 Arlington, Virginia 22217	1 Copy